instituto superior de engenharia de lisboa

Wi-Fi firmware for ESP32

Rafael Conceição
Eduardo Tavares

Orientadores: Diego Passos
Fernanda Passos

Relatório do projeto realizado no âmbito de Projecto e Seminário
Licenciatura em Engenharia Informática e de Computadores

Junho de 2025

Instituto Superior de Engenharia de Lisboa

Wi-Fi firmware for ESP32

41486 Rafael Gonçalo Figueiredo da Conceição

49454 Eduardo Dinis Gonçalves Tavares

Orientadores: Diego Gimenes Passos

Fernanda Gonçalves de Oliveira Passos

Relatório do projeto realizado no âmbito de Projecto e Seminário
Licenciatura em Engenharia Informática e de Computadores

Junho de 2025

Abstract

The goal of this work is to implement a medium access control (MAC) protocol through CSMA/CA (as used in IEEE 802.11) through software on a low cost environment, a characteristic which allows the protocol to be reconfigurable. This will be achieved by building firmware for Esp32 working as a Micro-controller Unit (MCU) for the CC1101 radio.

Besides the implementation of the protocol, this project includes a traffic generator responsible with periodically communication using the MAC, and optionally an application able to communicate with each network node, allowing the user to change their configuration.

The full implementation of this MAC protocol module will include the ability to use both RTS/CTS, a small exchange between nodes with a request to send data, and a confirmation response, along with the Network Allocation Vector (NAV), the mechanism which consists of announcing for how long the node predicts the communication will take in each frame sent. The client application naturally requires corresponding code on the Esp32 side, allowing for reconfiguration of network details during runtime.

The resulting product of this project should serve as an experimentation platform for the effectiveness of network platforms in a relatively low-cost environment.

Contents

1 Introduction **1**

1.1 Development reasons 1

1.2 Goals 1

1.3 Acronyms 2

1.4 Report Structure 2

2 Context **5**

2.1 Emulators 5

2.2 Hardware testbed testing 5

2.2.1 Software defined Radios 5

2.2.2 MCU systems 6

3 Organization **7**

3.1 Core 1 8

3.1.1 CC1101 and MCU 8

3.1.2 Traffic Generator 8

3.1.3 MAC protocol 9

3.1.4 CSMA/CA protocol 9

3.2 Application and Esp-32 11

4 Testing and Issues **13**

4.1 Buffer Overlfow in CC1101 Driver 13

5 Planning **15**

5.1 NAV 15

5.2 Application 15

5.2.1 Computer Application 15

5.2.2 Esp-32 Changes 16

Referências **18**

Chapter 1

Introduction

1.1 Development reasons

Wireless networks, as a technology increasingly being used in the modern world [1], are naturally studied to improve their efficacy. This study process will often contain tests as a way to study the effects of any new idea [2]. However, changed protocols cannot be run on computers, as the hardware itself does not allow this flexibility.

One possible choice for hardware would be software-defined radios [3], which, as the name suggests, can be configured through software. These can be capable enough to implement protocols following IEEE 802.11 standards, but their setback is the price of the equipment. The cost can reach thousands of dollars for a single set [4], something undesirable considering efficacy tests for a MAC protocol often involve a large amount of users attempting to use the network at the same time, due to concurrency of medium usage being a key factor.

This project intends to develop our own alternative using a CC1101 radio, the gateway to the wireless medium, controlled by a ESP32 [5] MCU, which will define the protocol at use at any time. The setback in this structure is the lack of ability to even try to reach the operational parameters of standard wireless networks, even at basic levels such as the bandwidth, along with the frequency and modulation of the signal.

Even if this solution has these limitations, this testbed is still useful as long as anyone using it considers them, and takes into account that lower frequencies may cause more interference due to them reaching longer distances, and the measurement of efficiency takes into consideration the lower bandwidth, avoiding direct comparisons such as measuring the useful amount of data sent per time.

1.2 Goals

The main goals of this work involve the development of:

- A MAC protocol primarily based on IEEE 802.11's Distributed Coordinated Function [6]. This will include an RTS/CTS mechanism as a way to help reduce the hidden node problem, as well as the NAV [7] as used in IEEE 802.11

- A traffic generator module, which will make use of the created MAC protocol and periodically send a package with a specific destination MAC address. Assuming the test is conducted as a high density network, sending data to only one specific node should not be an issue, as nodes would be close enough that any packet sent would occupy the medium for everyone
- Optionally, a computer application with the capability to communicate with each node. This would allow it to either modify the protocol used on the network or to collect data related with the experiment.

1.3 Acronyms

Some words and acronyms that will be used in this document:

- IEEE 802.11 - standard for wireless networks
- MCU/Micro-controller- a small device that works as a computer, capable of interacting with peripherals. Esp-32 takes this role on this project.
- MAC protocol - protocol devices should follow to access the medium
- NAV/Network Allocation Vector - process where devices attempt to reserve the medium by announcing for how long they assume they need the medium to communicate.
- CSMA/CA - basic idea for access to the shared medium, which involves listening to the medium and waiting until it is free.
- RTS/CTS - preamble to sending data where device asks permission from destination to start sending data.
- DCF/Distributed Coordination Function - most common and simple MAC protocol in IEEE 802.11.
- Driver - low level software to control hardware

1.4 Report Structure

The structure of this report, by chapter, is as follows:

- Chapter 2: dedicated to discuss the current tools used to tackle the driving problem introduced in section 1.1
- Chapter 3: explains the structure chosen to support the whole project, including the optional goal, along with the current code organization

- Chapter 4: covers issues found during this project, along with the testing strategy on MCU time bounded software

Chapter 2

Context

This chapter discusses the environment surrounding this project, focusing on already existing ways to test network protocols.

2.1 Emulators

One existing way to test the reliability of networks is the usage of emulators, such as the Network Simulator 3 [8]. The utilization of emulators allows for the emulation of a network with multiple nodes from a single device, all with a custom MAC.

This method, although useful to have a good idea of the tested protocol's effectiveness, tends to not be completely accurate. By its very nature, it tends to idealize and abstract certain aspects of the network. This makes testing using actual hardware desirable, even if it entails the cost of acquiring such hardware. Another downside is the energy inefficiency of emulators, which need to run on a more powerful machine - it must simulate all the nodes on top of a fully functioning machine and Operating System.

2.2 Hardware testbed testing

2.2.1 Software defined Radios

As mentioned in the introduction, the usage of software defined radios (SDR solutions) would be the implementation most capable alternative to reach the standards of normal WiFi as defined by IEEE 802.11. This capability to replicate real WiFi behavior would make for the easiest analysis of the test results, due to not needing to account for the differences in the physical layer, as these differences can lead to unexpected results even if testing known protocols.

The primary issue with this method is cost. Some of the equipment needed can cost hundreds of euros at best, and tens of thousands of euros at the most expensive [3]. The cheapest solution we could find with a quick search at the time of this writing was the "SPECTRAN® V6 ECO 150XA-6", a product by Aaronia, which costs 3.498€ [9].

2.2.2 MCU systems

An MCU system, as used in this work, is the usage of an MCU as a way to control a simple, low capacity radio.

Although an equivalent system to the one being developed for this project does not appear to exist, the implementation of IEEE 802.15.4 was created [10], however, that is for an STM32 architecture, not an ESP32.

Chapter 3

Organization

As we can see in Figure 3.1, this project is structured to leverage the dual-core architecture of the ESP32 MCU, with each core assigned specific responsibilities to optimize performance. This division is essential for effectively managing the various factors that affect the system operation.

The reasoning behind this organizational approach is simple: certain operations, such as handling large data packets, necessitate uninterrupted collaboration from the MCU. In addition, the communication protocols used are subject to strict timing requirements[1]. By dedicating one core to manage the radio communication continuously, while the other handles the future client app (not yet implemented), we can ensure that these constraints are met without compromising the overall system performance.

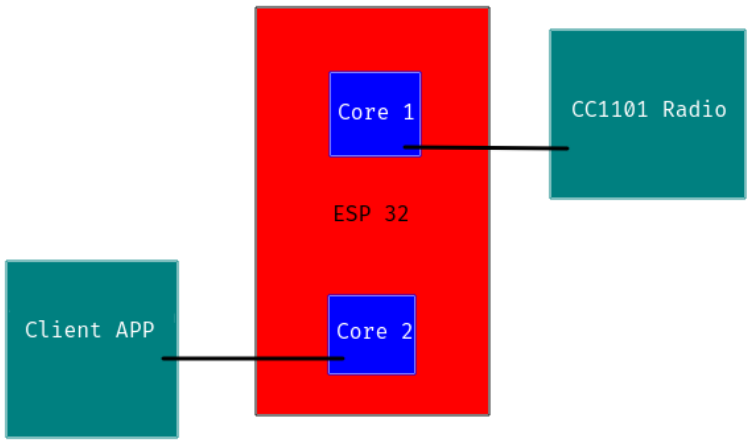


Figure 3.1: Esp core tasks

[1]This mostly refers to acknowledging data frames and RTS, which expect an answer under SIFS (Short Inter-frame Space) constraints.

3.1 Core 1

The current organization of the code on core 1 is shown in Figure 3.2. This should be loosely interpreted as a dependency graph between classes, although sometimes classes do not actually contain an instance of the dependency (instead opting to receive a pointer to the desired functions), and the MAC protocol does not have a corresponding class.

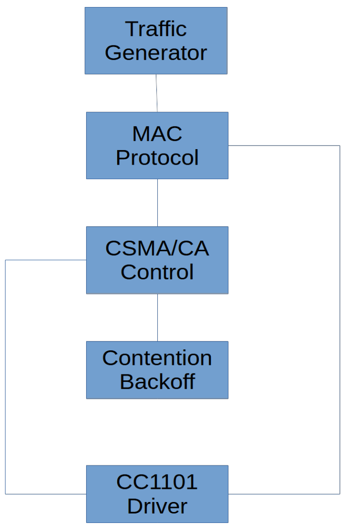


Figure 3.2: Esp core tasks

3.1.1 CC1101 and MCU

The CC1101 driver is achieved using an already implemented solution, provided to us by our advisors, to communicate with the radio module.

Communication between the CC1101 [11] and the MCU is done through the Serial Peripheral Interface (SPI) [12] standard. This protocol is used to read values from the radio and change its operational status.

In addition to SPI communication, the CC1101 utilizes two General Digital Output (GDO) pins, GDO0 and GDO2. Each of these pins outputs a single-bit value that the MCU can read. The significance of these values is determined by the specific configuration settings applied to the radio, which are adjusted by manipulating the registers through SPI communication.

3.1.2 Traffic Generator

The traffic generator is responsible for creating packets based on specified parameters, including the source and destination addresses. It also accepts a time parameter that determines the interval between generated packets.

Radio pin number	Name	Purpose	Esp pin name
1	GND	Ground	GND
2	VCC	Energy connection	3V3
3	GDO0	Bit sign	D4
4	CSN	SPI connection	D5
5	SCK	SPI connection	D18
6	MOSI	SPI connection	D23
7	MISO	SPI connection	D19
8	GDO2	Bit sign	D2

Table 3.1: Female jumper cable connection

The time interval can be set to either a constant value or follow a normal distribution; in the latter case, the provided value will represent the mean of the distribution. By default, and if no type is mentioned, the interval will be constant.

The message contained in the packet is for now a constant string, containing every character in the alphabet and the numbers 0 to 9, repeated.

3.1.3 MAC protocol

The MAC protocol serves two primary responsibilities: it creates the send function utilized by the traffic generator, and allows listening for incoming messages while not actively transmitting data. This dual functionality is essential, as positive acknowledgments are a cornerstone of this protocol, and the NAV protocol [13] requires the system to monitor packets that are not directed to the local node itself.

To support these operations, this module delegates the logic of waiting for its turn to the CSMA Control class, which manages access to the shared communication medium. It also interfaces with the radio driver to facilitate both data transmission and reception.

Given that communication initiated by other devices is inherently asynchronous, it is not feasible to handle these interactions within a single code thread. To address this challenge, we have opted to use FreeRTOS [14], a lightweight real-time operating system. When the system begins to receive a packet, the driver interface generates an interrupt signal that serves as a notification. The protocol implementation, when not actively expecting a packet, will resume a FreeRTOS task specifically designed to handle incoming packets. This task will verify both the address and the duration field[2] of the incoming packet, responding as necessary.

3.1.4 CSMA/CA protocol

The objective of this class is to ensure the protocol follows the wait time shown in Figure 3.3. To achieve this goal, the radio driver will need to receive a method to perform a clear channel assessment (cca). Such a method will be used to determine if the channel is free for either DIFS or backoff amount of time, depending on which is needed.

[2]This field is part of NAV, indicating in microseconds for how long it plans to use the medium.

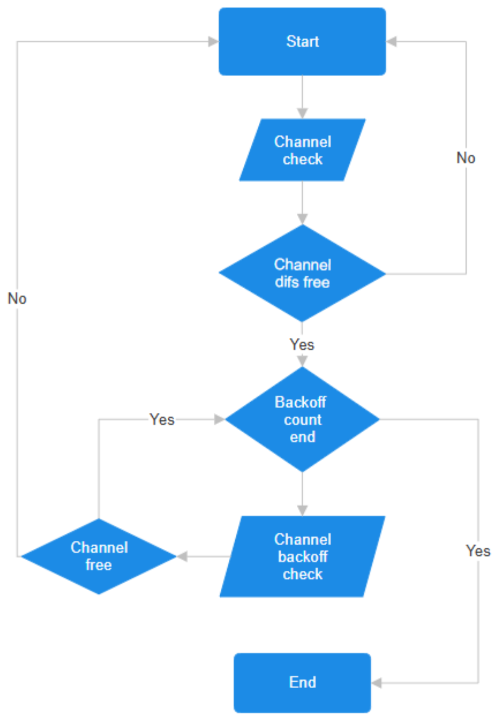


Figure 3.3: CSMA wait for turn flowchart

During the implementation of the checks for those time slots, it was decided to call the cca function repeatedly to cover the entire duration, reading a radio status register repeatedly from the C1101 using SPI. If the driver changes the determination of cca by setting the GDO2 pin to expose the carrier sense, the implementation could be changed to sleep for the time slot duration, waking up through an interruption if the pin value is high. This change could make it easier to measure for the required amount of time, as the current method of reading from a register may be too slow, leading to extra time needed to cover the whole slot duration.

Contention Back-off

The ContentionBackoff class is a simple interface, whose implementations will dictate the rules upon which the contention window is determined.

The contention window is an integer number that can be redefined depending on whether the communication was successful or not. By defining the functions reduceContentionWindow and increaseContentionWindow, various behaviors, such as Multiplicative Increase, Linear Decrease, Exponential Backoff (as used in DCF), or even a constant window can be easily implemented and replaced.

3.2 Application and Esp-32

Although the production of this module has not yet begun, it is expected that the communication will be built upon the actual Esp-32 WiFi capabilities, which is supported completely apart from any MAC protocol being tested.

This module would require parsing the information received from the application and changing the protocol parameters as requested. If this change includes changing any parameter of the radio, such as switching frequencies, it should be done by either pausing any task running on core 1, or by creating a task to run on core 1 that will do those changes, as the risk of attempting to communicate with the radio on 2 cores at once could cause unexpected bugs.

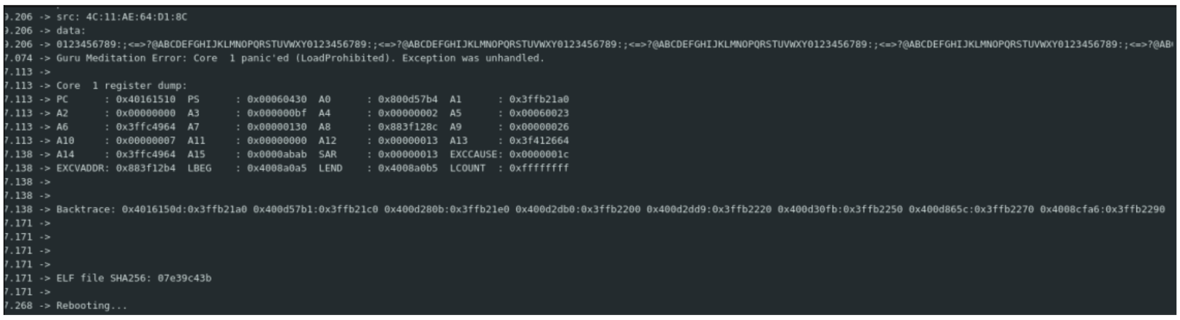
Chapter 4

Testing and Issues

This chapter we will cover some of the issues we faced. Our tests were mostly conducted using the Serial Monitor – this is a way for the micro controller to send messages over USB – as a way to print messages and check the state of the system, and see wether or not it receives messages. One setback we face with this and which we must take into account is how slow the communication is. This can provoke delays which, given the strict timing of the system and protocols, can cause issues. Another thing we must account for is how (if we are connecting both nodes to one laptop) we can only monitor the output of one node at a time.

4.1 Buffer Overlfow in CC1101 Driver

One issue we faced was the seemingly random crashing of the nodes. At unpredictable and apparently random times, one or two of the nodes would simply cease to function. We eventually managed to capture one of the errors, which can be seen in Figure 4.1.



```
.206 -> src: 4C:11:AE:64:D1:8C
.206 -> data:
.206 -> 0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXY0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXY0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXY0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXY0123456789:;<=>?@AB
.074 -> Guru Meditation Error: Core  1 panic'ed (LoadProhibited). Exception was unhandled.
.113 ->
.113 -> Core  1 register dump:
.113 -> PC      : 0x40161510  PS      : 0x00060430  A0      : 0x800d57b4  A1      : 0x3ffb21a0
.113 -> A2      : 0x00000000  A3      : 0x000000bf  A4      : 0x00000002  A5      : 0x00060023
.113 -> A6      : 0x3ffc4964  A7      : 0x00000130  A8      : 0x883f128c  A9      : 0x00000026
.113 -> A10     : 0x00000007  A11     : 0x00000000  A12     : 0x00000013  A13     : 0x3f412664
.138 -> A14     : 0x3ffc4964  A15     : 0x0000abab  SAR     : 0x00000013  EXCCAUSE: 0x0000001c
.138 -> EXCVADDR: 0x883f12b4  LBEG    : 0x4008a0a5  LEND    : 0x4008a0b5  LCOUNT  : 0xffffffff
.138 ->
.138 ->
.138 -> Backtrace: 0x4016150d:0x3ffb21a0 0x400d57b1:0x3ffb21c0 0x400d280b:0x3ffb21e0 0x400d2db0:0x3ffb2200 0x400d2dd9:0x3ffb2220 0x400d30fb:0x3ffb2250 0x400d865c:0x3ffb2270 0x4008cfa6:0x3ffb2290
.171 ->
.171 ->
.171 ->
.171 ->
.171 -> ELF file SHA256: 07e39c43b
.171 ->
.268 -> Rebooting...
```

Figure 4.1: Radio crash

To find the root of this issue, we copied the "Backtrace" hex values into a text file called "backtrace_addr" separated by lines, and, together with the *.elf* file, we used the following bash script to decode the error:

```
#!/bin/bash
```

```
while read addr; do
    addr2line -e wifi_firmware_esp32.ino.elf "$addr"
done < backtrace_addr
```

This script pointed to the error taking place inside an SPI function, which is outside the domain of this project. This issue was communicate to driver developer, who discovered the cause was potentially due to the radio firmware receiving a packet that seemed to exceed the max size that had been defined. This could potentially occur due to a corruption issue during the transmission of the packet.

Chapter 5

Planning

In this chapter it´s explained what capabilities are planned to being added, along with a prediction of what challenges such changes will entail

5.1 NAV

The addition of NAV will require adding a duration field to every frame sent, indicating for how long the node desires to acquire exclusive access of the medium to accomplish a atomic communication.

Since the duration of communication will vary depending on the amount of data bits to send, the duration fields in the RTS/CTS frames, which happen before the data transmission, will be affected. This may lead to changing the code organization, since the data frames and the RTS/CTS frames are being dealt by different modules of the code (by the traffic generator and the MAC protocol respectively).

When adding it to the MAC protocol, it will consist of reading the duration field and wait the rad amount of microseconds.

5.2 Application

The previously mentioned configuration application will require the development of two modules; the actual application code for the device, and the Esp-32 code ready to recept the data started by the application.

5.2.1 Computer Application

This application´s interface will consist of text based commands, such as the ones found accessible through a shell program. The main goals of this application will be to allow to configure:

- the destination address for the communication
- the traffic generator time interval

- the choice of contention back-off algorithm

The possible configuration choices for things such as the destination address and back-off algorithms will be limited by the existing addresses and implemented back-off algorithms. This application will likely be configured to know the possible choices statically. Such a method will require recompiling the source code with every change.

5.2.2 Esp-32 Changes

The added code will start with parsing any received information from the application. As was explained in Chapter 3, this will be done on a separate core to the rest of the code, as a way to minimize the impact this addition would have on the traffic generator and MAC protocol.

An expected challenge of this section would be dealing with the synchronization of memory between the two cores. Since each core will have their own cache, among other optimizations, changing memory shared between cores may not be straightforward [15]. There are two main ideas to solve this issue:

1. Use both volatile flags and other synchronization techniques provided by freeRtos (such as a semaphore, mutex, or queue)
2. avoid this issue delegating the memory change to a task ran on the same core. Task creation in freeRtos allows adding a parameter to the task function, which would be retrieved information from the app on this solution.

Bibliography

[1] Cisco staff. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html. cisco internet anual report.

[2] Dave Cavalcanti, Carlos Cordeiro, Malcolm Smith, and Alon Regev. Wifi tsn: Enabling deterministic wireless connectivity over 802.11. *IEEE Communications Standards Magazine*, 6:22–29, 12 2022.

[3] W.H.W. Tuttlebee. Software-defined radio: facets of a developing technology. *IEEE Personal Communications*, 6(2):38–44, 1999.

[4] Wikipedia contributors. Sdr list. https://en.wikipedia.org/wiki/List_of_software-defined_radios.

[5] Esp 32 description. https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32/esp32-devkitc/user_guide.html#functional-description.

[6] Wikipedia contributors. Distributed coordination function. https://en.wikipedia.org/wiki/Distributed_coordination_function.

[7] Wikipedia contributors. Network allocation vector. https://en.wikipedia.org/wiki/Network_allocation_vector.

[8] https://www.nsnam.org/documentation/. ns3 documentation.

[9] Aaronia. Spectranv6. https://aaronia.com/en/shop/spectrum-analyzer/spectran-v6-eco-150xa-6.

[10] Wojciech Domski. Sx1278-example. https://github.com/wdomski/SX1278-example/blob/master/README.md.

[11] https://www.ti.com/lit/ds/symlink/cc1101.pdf. CC1101 datasheet.

[12] Wikipedia contributors. https://en.wikipedia.org/wiki/Serial_Peripheral_Interface. spi explanation.

[13] Matthew Gast. *802.11® Wireless Networks: The Definitive Guide*. O'Reilly, April 2002. defining nav duration explained through chapter 4.

[14] https://www.freertos.org/Documentation/01-FreeRTOS-quick-start/01-Beginners-guide/01-RTOS-fundamentals. FreeRTOS fundamentals.

[15] https://medium.com/@pkgmalinda/understanding-synchronization-issues-in-multi-threaded- multi thread memory synchronization example.