UCLouvain

École polytechnique de Louvain

6LoWPAN for UWB communication on the GRiSP 2

Implementation and Evaluation

Author: **Kouassi Jonathan Affrye**
Supervisor: **Peter Van Roy**
Readers: **Ramin Sadre, Peer Stritzinger, Gwendal Laurent**
Academic year 2023–2024
Master [120] in Computer Science and Engineering

Abstract

Connected objects have become an integral part of our society, combining various technologies and providing a wide range of applications. One of these applications is real-time object tracking. While several technologies can achieve this, UWB (Ultra-Wideband) stands out for its superior performance in indoor environments where high precision is required. However, the true appeal of connected objects lies in their ability to exchange information over the Internet, enabling users to control and monitor their applications remotely.

This thesis focuses on the implementation of the 6LoWPAN protocol on the GRiSP 2 embedded system, which is equipped with a UWB sensor. The 6LoWPAN protocol is crucial as it bridges the gap between the Internet and IoT devices, allowing these devices to benefit from the advantages of internet connectivity. To achieve this, the work consisted of implementing the features defined in RFC 4944 and RFC 6282, through the design and implementation phases, as well as software and hardware tests to enable the exchange of IPv6 packets on the GRiSP 2 board using UWB. The results of the routing, compression and fragmentation tests were conclusive, paving the way for implementation of the upper layers.

Acknowledgements

First of all, I would like to thank Mr. Peter Van Roy for giving me the opportunity to work on this thesis and for his guidance throughout the year.

A huge thank you to Gwendal Laurent, who played the role of tutor and advisor throughout the year. I am grateful for his availability, insightful reflections, and valuable suggestions during every phase of this thesis, from implementation to writing.

A special thanks to Peer Stritzinger for his trust.

I would also like to thank Mr. Jean-Didier Legat, who played a decisive role in allowing me to continue my studies while presiding as the chairman of the EPL jury. I am glad that his decision was not in vain.

I also want to acknowledge all the professors I had throughout my studies and the tutors assigned to us. Many of them were a driving force for me, inspiring me and making me want to learn more.

Among these professors, I would particularly like to thank Mr. Ramin Sadre Sadre and Mrs. Cristel Pelsser, whose courses on embedded systems and mobile technologies awakened my passion in this field and were especially helpful in this thesis.

Finally, I would like to express my deep gratitude to my family and friends, who have constantly encouraged me throughout my studies, through their words and actions, both in the good times and in moments of doubt and self-reflection.

All the glory to God!

Contents

List of Figures

List of Tables

Acronyms

BC Broadcast

CID Context Identification

CSMA/CA Carrier Sense Multiple Access/ Collision Avoidance

DP Destination Port

DAC Destination Address Compression

DAM Destination Address Mode

DCI Destination Context Identifier

DSCP Differentiated Services Code Point

ECN Explicit Congestion Notification

HLIM Hop Limit

ICMP Internet Control Message Protocol

IEEE Institute of Electrical and Electronics Engineers

IID Interface Identifier

IP Internet Protocol

IPHC Internet Protocol Header Compression

IoT Internet of Things

LRWAN Low-rate wireless area networks

MAC Medium Access Control

MTU Maximum transmission Unit

NALP Not a Lowpan Packet

NHC Next Header Compression

NH Next Header

PAN Personal Area Network

RFC Request for comments

RTLS Real-Time Location Systems

SAC Source Address Compression

SP Source Port

SAM Source Address Mode

SCI Source Context Identifier

TCP Transmission Control Protocol

TF Traffic Class

UDP User Datagram Protocol

UWB Ultra wide band

WPAN Wireless personal area networks

Chapter 1

Introduction

In the 1990s, the term "Internet of Things" (IoT) began to emerge, referring to a collection of devices designed to perform specific tasks. These devices, connected over wired or wireless networks and communicating through various protocols, are equipped with sensors, actuators and other components that allow them to meet a variety of needs. IoT has become a major component of our society, enhancing our quality of life. In 2018, approximately 6.1 billion devices were connected to the Internet. Initial expectations were that the number of connected devices would reach around 8.9 billion by 2020, [3] today, these numbers have not only been reached but continue to grow, with estimates reaching 32 billion of connected devices by 2030. [4]

However, as promising as they may be, IoT devices belong to a class known as low power and low rate device, such devices come with various constraints, starting with the limited processing power, low memory, low power, low data rate. Additionally, networks equipped with such devices, are interconnected through lossy links, which are unreliable and can make communication between nodes difficult when fails occur. [5]

All these constraints make the deployment of the Internet complicated on low-rate wireless area networks (LRWAN). The publication of the IEEE 802.15.4 standard was a significant step in this direction, paving the way for the adaptation of IP on such networks. [6] However, using the internet implies the use of an IP protocol such as IPv4 or IPv6. Given the address space limitations and security concerns of IPv4 [7] more and more IoT systems are adopting IPv6. The latter impose a maximum transmission unit (MTU) of 1280 bytes [8] which is the largest packet size the IP layer can transmit without the need for fragmentation [9]. In contrast, the IEEE 802.15.4 defines an MTU of 127 bytes [10] making it impossible to transmit IPv6 packets over low-power devices using the IEEE 802.15.4 protocol at maximum size without an adaptation layer. In addition, IP introduces several challenges when deployed on LoWPAN networks, such as the need for devices to auto-configure their addresses in a stateless manner, manage mesh topologies, where multiple hops are required and intermediate devices act as packet relays.

To address these issues, the 6LoWPAN standard was developed. 6LoWPAN stands for IPv6 over Low-Power Wireless Personal Area Networks and allows the use of IPv6 on wireless networks with low power consumption and low data rates. As shown in the figure 1.1, who represents the general 6LoWPAN stack, 6LoWPAN is located just above the IEEE 802.15.4 Mac layer and below the network layer.

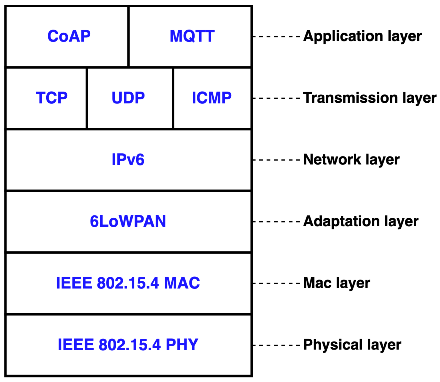


Figure 1.1: 6LoWPAN stack

1.1 6LoWPAN features

To benefits from advantages of IPv6, 6LoWPAN comes with several mechanisms.

The first mechanism is the IPv6 header compression. It is based on the principle that some header field values can be inferred or omitted based on the network context, thus reducing the size of the packet that should be transmitted. Header fields of IEEE 802.15.4-2011 packets leave very little space for actual data, from 88 to 102 bytes depending on the security options and addressing type as shown in the following figure. [11]

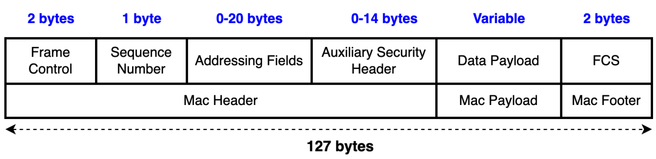


Figure 1.2: IEEE 802.15.4-2011 mac header

Using the compression mechanism the header size can significantly be reduced, going from 48 bytes to 6 bytes in the best case scenario. [12]

The second mechanism is packet fragmentation, which ensures that application layers can operate without needing to consider the physical layer's transmission constraints. This allows large packets to be divided into smaller fragments at the 6LoWPAN layer and then reassembled later.

The final mechanism considered in this work is mesh forwarding. One of the challenges of IP on IEEE 802.15.4 is to enable link-layer routing within a mesh network topology, which allows each network node to act like a router. Given that losses are frequent and links quality are not always guaranteed, it is important to develop a robust mechanism for efficiently delivering packets from the initiator to the recipient.

1.2 Thesis context

Most embedded systems today must be programmed in low level languages like the C programming language, but those languages are not always easy to begin with, they require some skills in order to avoid inappropriate usage of device resources and some core functions are known to contain security vulnerabilities. To avoid these constraints, the German company, Peer Stritzinger GmbH has created the GRiSP 2 board. It is an embedded system, developed in the Erlang ecosystem, enabling the design of IoT applications "out of the box", thus simplifying the development process for embedded systems. [13]

In this work, we focus on the integration of Ultra-Wide band (UWB) technology on the GRiSP 2 board. UWB is a wireless communication technology that uses pulse transmissions to enable information exchange between two devices. It is particularly advantageous for real-time location and tracking due to its higher precision compared to ultrasonic sensors or GPS signal. [14] Companies like Apple, with their AirTags, and Samsung use UWB for precise device localization. [15]

1.3 Contributions

This thesis makes two major contributions:

- First is the implementation of compression, fragmentation, meshing mechanisms and the stateless address auto-generation as defined in the 6LoWPAN standard. This step required designing an appropriate code architecture to work within the Erlang programming language and defining an API that would allow both the MAC layer and the IP layer to communicate with the 6LoWPAN layer.

- Second is the establishment of a set of tests to validate the implementation and enable the GRiSP 2 boards to exchange IPv6 packets while utilizing 6LoWPAN's features, particularly meshing. At this stage, the boards can already exchange data via the Pmod sensors, but no routing mechanism has been implemented. Implementing routing is a crucial step towards enabling data flow within a network of GRiSP boards.

This work is part of the broader vision of integrating the Thread standard into GRiSP boards. Thread is an IPv6-based networking protocol designed for low-power IoT devices in an wireless mesh network. [16]

The complete code produced for this thesis, along with the code from previous work, can be found in the appendix and is accessible on GitHub via the following link [17].

1.4 Document structure

This document is organized as follows, Chapter 2 discusses the current state of technology that led to the use of a UWB as well as the needs of an adaptation layer. Chapter 3 details the hardware used in this thesis, along with their characteristics. Chapter 4 reviews previous work that has been done, which served as a foundation for this thesis. In Chapter 5, we delve into the core subject by discussing the 6LoWPAN protocol, followed by Chapter 6, which covers how it was implemented in Erlang. Chapter 7 presents the test performed and results obtained for both simulation and real life cases. Chapter 8 outlines the limitations of the current implementation and suggests possible improvements. Finally, Chapter 9 concludes with a summary of the key points discussed and the results of this work.

Chapter 2

State of the art

This chapter reviews key technologies related to this thesis, focusing on the development of the 6LoWPAN protocol for embedded systems using UWB technology.

It begins with an exploration of localization techniques and technologies in embedded systems, emphasizing why UWB is particularly promising. Then, it examines the benefits of using Erlang as a programming language for embedded systems. Lastly, it discusses communication protocols for deploying embedded applications and why 6LoWPAN is the preferred choice.

2.1 Localization techniques and UWB

Localization technologies offer significant potential for IoT systems, particularly in applications where precise real-time location tracking is essential, such as in industrial automation or healthcare. Various techniques have been developed, each finding its utility in specific applications. Here, they will be discussed with a focus on accuracy and low latency.

2.1.1 WiFi and BLE

Wi-Fi, operating at 2.4 GHz or 5 GHz, offers accuracy ranging from 0.4 to 5 meters, making it well-suited for large indoor environments like warehouses [18]. BLE, also operating at 2.4 GHz, provides slightly better precision with an accuracy of 0.9 to 2 meters, which is ideal for applications such as asset tracking in hospitals [18]. However, both Wi-Fi and BLE face challenges with latency and environmental interference, which limits their effectiveness in high-precision, real-time localization systems.

2.1.2 Zigbee and Acoustic Ultrasound

Zigbee, commonly used in low-power IoT applications, operates at 2.4 GHz and provides localization accuracy ranging from 0.8 to 5 meters [18]. While effective in smart home environments, its suitability diminishes in scenarios that require real-time precision [19].

Acoustic ultrasound, operating at 20 kHz, offers higher accuracy between 0.1 to 0.6 meters. However, its performance is hindered by environmental noise and its limited range, making it less practical for large-scale deployments [18].

2.1.3 Global Navigation Satellite System (GNSS)

Global Navigation Satellite Systems, such as GPS, operate on Radio Frequency and are highly effective for outdoor localization, offering accuracy between 3 to 15 meters [18]. However, their performance significantly degrades indoors due to signal attenuation and multipath effects, making GNSS less suitable for environments where high precision and low-latency tracking are critical.

2.1.4 Ultra-Wideband (UWB)

Ultra-Wideband is a robust option for indoor localization, particularly in scenarios demanding high precision and low latency. Operating across a wide frequency range (3.1 GHz to 10.6 GHz), UWB typically offers accuracy between 0.3 to 0.5 meters, with certain systems achieving precision down to a few centimeters [18]. UWB's resilience to multipath effects and its low-latency communication make it ideal for real-time tracking in environments like industrial automation and healthcare [19].

2.2 Embedded systems programming

Various programming languages are employed in embedded systems and IoT, each with its strengths and limitations. C and C++ are commonly used due to their low-level hardware access and high performance, making them indispensable for resource-constrained environments. However, these languages require developers to manually manage concurrency and fault tolerance, often resulting in complex and error-prone code.

Python, while praised for its simplicity and extensive libraries, can struggle with performance limitations, especially in highly concurrent or real-time systems. [20]

Java provides better concurrency support but introduces latency through garbage collection, which is undesirable in real-time applications. [21]

Rust, has gained popularity recently for its emphasis on memory safety and performance, largely due to its unique ownership model. It provides the low-level control seen in C/C++ while offering protections against common programming errors like null pointer dereferencing and data races. [22] However, Rust's steeper learning curve and still-maturing ecosystem can pose challenges for developers new to the language or working in highly specialized domains.

In contrast, Erlang is designed for building highly concurrent, distributed, and fault-tolerant systems. Originally developed for telecommunications, it excels in environments where systems must remain operational despite frequent failures. Erlang's support for managing thousands of lightweight processes, hot code swapping, and distributed computing makes it ideal for IoT applications requiring high availability, scalability, and ease of maintenance [23] [24].

2.3 Communication protocols and 6LoWPAN

In the IoT landscape, choosing the right communication protocol is essential for balancing power consumption, range, data rate, and network scalability. When considering the Thread standard, which is designed for secure, scalable, and reliable mesh networks in IoT environments, 6LoWPAN emerges as the preferred choice over alternatives like Zigbee, Z-Wave, BLE, and Wi-Fi.

Zigbee is known for its low power consumption and robust mesh networking capabilities, making it a popular choice in smart home automation and industrial control. Z-Wave, operating in the sub-1 GHz band, provides greater range and reduced interference, though it is proprietary and region-dependent. BLE, favored for its low energy use and integration with mobile devices, excels in applications requiring interaction with smartphones but falls short in range and data throughput for more demanding deployments. Wi-Fi, despite its widespread use and IP-based structure, is not optimized for low-power operations, making it less suitable for battery-operated IoT devices.

6LoWPAN, on the other hand, offers native support for IPv6, enabling seamless integration with existing internet infrastructure. It operates efficiently over IEEE 802.15.4 networks, providing data rates up to 250 kbps, ideal for low-power, battery-operated IoT devices. The protocol's support for mesh networking enhances the reliability and scalability of IoT networks, aligning perfectly with the goals of the Thread standard.

Chapter 3

Hardware description

This chapter presents the equipment and sensors used in this thesis and their characteristics. The equipment consisted of the GRiSP2, a Pmod UWB sensor and a UWB sniffer.

3.1 GRiSP 2 board

The GRiSP board, developped by Peer Stritzinger GmbH, is an embedded system designed for high-level application development using Erlang. The board uses the RTEMS real-time operating system, which allows Erlang application to run directly on the hardware without an intermediary layer, facilitating real-time and high-level applications and simplifying the development process. The board includes Wi-Fi and USB connectivity, making it ideal for IoT applications. It also supports various expansion modules called Pmods, which offer a range of sensors and actuators. Grisp board provides a complete Erlang Virtual Marchine (VM) and supports Elixir via Nerves and Linux. Users can interact with the board through an Erlang shell accessible via serial or Wi-Fi connection. [25]

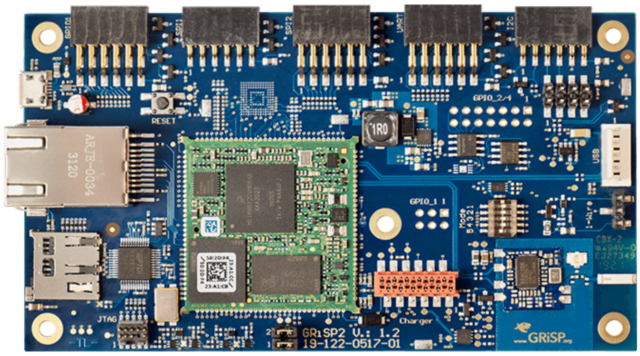


Figure 3.1: GRiSP 2 board [1]

Compare to a previous version of the GRiSP board, GRiSP 2 features more powerful CPU, enhanced IO throughput, an Ethernet port. It comes with a complete toolchain for embedded project development.

3.2 Pmod sensor - DW1000

Pmod modules are small and low-cost peripheral modules such as sensors or actuators designed to extend the functionalities of an embedded system. [26]. The Pmod DW1000 is a sensor module, that enables accurate distance measurement within 10 centimeters, making it ideal for applications requiring exact positioning. It utilizes UWB technology and complies with the IEEE 802.15.4-2011 standard, defining the physical and MAC layers for low-rate wireless personal area networks (LR-WPANs). [27]

The module connects to the GRiSP board through a 12-pin SPI interface, facilitating high-speed data exchange. It operates on multiple channels with center frequencies ranging from **3494.4 MHz** to **6489.6 MHz**, enhancing its robustness and flexibility. With an extended communication range of up to **250 meters** at **110 kbps**, the Pmod DW1000 is suitable for real-time location systems (RTLS), indoor navigation, asset tracking, and applications requiring concurrent data transfer and precision location. [27]

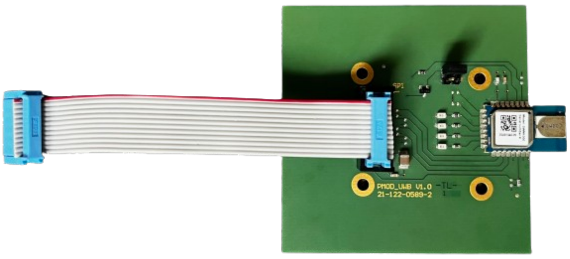


Figure 3.2: Pmod uwb

3.3 UWB sniffer

The UWB sniffer is aimed for debugging precise RTLS and active RFID systems. It is integrated with the industry opensource software, Wireshark, supports six channels (802.15.4a UWB PHY) up to **6.5GHz** and features an Ethernet communication interface. The device allows easy automation via an HTTP interface and provides the received signal strength indication (RSSI). [28]

It offers two main operation modes: Sniffing, which captures UWB frames and forwards them to Wireshark, and Injection, which allows users to send custom UWB frames from a web interface, making it ideal for device development, testing, and auditing. [28]

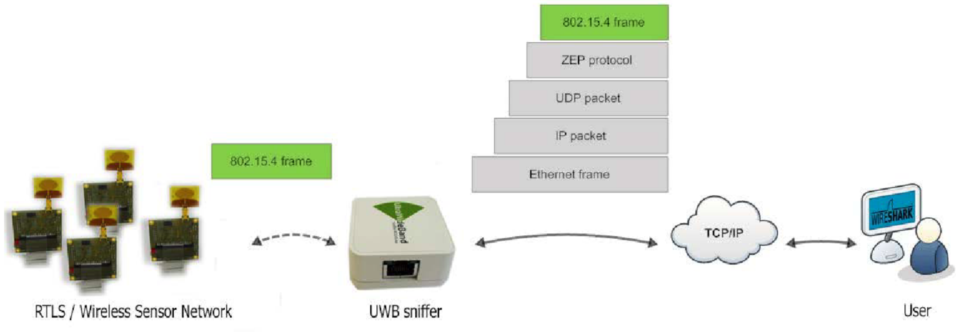


Figure 3.3: UWB Sniffer operation [2]

The UWB Sniffer also supports dissecting the Decawave Two Way Ranging protocol. In order to operate, the device requires Wireshark software, a USB port or DC adapter, and an Ethernet port. Configuration and usage involve setting up the network and connecting to the device's homepage for specific operations. [28]

Chapter 4

Previous work

The 6LoWPAN layer builds on previous work done by Gwendal Laurent during his master thesis. [29] This work was divided into two phases: first, developing a driver to support the new Pmod UWB sensor developed by the company Peer Stritzinger GmbH, based on the DW1000; and second, implementing the Medium Access Control (MAC) layer above the driver. The final objective was to send and receive data frames complying with the IEEE802.15.4-2011 standard.

The driver was developed to ensure communication between the Pmod UWB and the GRiSP 2 board, utilizing the DW1000's SPI interface, which facilitates communication between micro-controllers and peripheral devices, allowing fast data sharing between a master and one or several slaves via a synchronous serial connection. The DW1000 also offers MAC layer features such as frame filtering, CRC generation, checking, and automatic acknowledgment. The implementation focused on encoding and decoding the MAC header and managing MAC frame transmission and reception.

Additionally, distance measurements between nodes were conducted using two methods based on the concept of two-way ranging: Single-sided Two-way Ranging and Double-sided Two-way Ranging . These measurements demonstrated that the boards are capable of transmitting at a rate of 32.44 kb/s, and the distance calculation methods produced better than expected results. [29] By enabling communication between two GRiSP boards, this work provides a solid foundation for the development of real-time location system applications.

4.1 Mac layer usage and API

Gwendal has continued to develop the MAC and physical layers in parallel with this thesis in order to provide an API for the 6LoWPAN layer, enabling the transmission of datagrams to the MAC layer and the reception of frames from the MAC layer. The most useful API functions for our implementation are listed below.

- **set_pib_attribute** : This function is used to set the value of a PIB attribute. Those are attributes required to manage the MAC sub-layer of a device. In this work, it was primarily used to define the MAC address, either 16 bits or 64 bits.

```
-spec set_pib_attribute(Attribute, Value) -> ok when
      Attribute :: pib_attribute(), Value     :: term().
```

- **start**: This function starts the process in charge of the IEEE 802.15.4 layer. It takes one parameter, a map that defines the modules implemented by the physical layer (used only for simulation), the module implementing the duty cycle, and the callback function used for data reception and processing on the MAC layer.

```
-spec start(Params) -> {ok, pid()} | {error, any()} when
    Params :: map().
```

- **transmission**: This function is used to transmit datagrams on the MAC layer. It takes one parameter, a tuple containing a frame control record, a mac header record and the actual data in bitstring format.

```
-spec transmission(Frame) -> {ok, map()} | {error, Error} when
      Frame         :: {map(), map(), bitstring()},
```

- **rx_on**: This function activates the reception mode of the physical layer. In this mode, the boards is ready to receive data and transmit it for further processing.

```
-spec rx_on() -> ok | {error, atom()}
```

Chapter 5

6LoWPAN

This chapter describes the features of the 6LoWPAN standard that have been considered as part of this work. It begins with an overview of the 6LoWPAN, then covers what is defined in RFC4944 and RFC6282 which served as a reference for the implementation. In this document, 'Node' and 'device' are used interchangeably to refer a physical device in the network.

5.1 Overview

The 6LoWPAN protocol, developed by the IETF's 6LoWPAN working group, was established to address the issue of transmitting IPv6 datagrams over wireless networks with low energy consumption and low data rates [30]. In 2007, RFC 4919, described Low-power Wireless Personal Area Networks (LoWPANs), detailing their IPv6 integration, IP layer requirements, and network communication challenges."[6]. Later that year, RFC 4944 discussed the format of 6LoWPAN packets for IPv6 datagram transmission, including stateless auto-configuration of IEEE 802.15.4 addresses, a simple header compression scheme, and the way packets should be routed over the IEEE 802.15.4 MAC layer [31]. This RFC was updated by RFC 6282 in 2011, which introduced a new compression scheme of IPv6 header, UDP next header and a mapping for multicast addresses.[32]. Additional standards were developed over the years, such as RFC 6568 (2012), which explored the design space dimensions for LoWPAN applications [5], and RFC 6606 (2012), which addressed the lack of mesh topology specifications in IEEE 802.15.4 and 6LoWPAN, offering routing guidelines [33]. Finally, RFC 6775 (2012) optimized neighbor discovery for 6LoWPAN networks, improving addressing mechanisms and duplicate address detection [34].

5.2 RFC 4944

In this section, we will look at the features described in RFC 4944 that have been implemented in this work, namely, datagram encapsulation, meshing, fragmentation, stateless address auto-configuration as well as multicast address mapping.

5.2.1 IEEE 802.15.4 mode for IP

The 2011 IEEE standard specifies four frame types: beacon frames, MAC command frames, acknowledgment frames, and data frames. To enable link-layer recovery, transmission of IPv6 datagrams should utilize frames that require acknowledgments.The RFC allows for IEEE 802.15.4-2011 networks to operate in either beacon or non-beacon modes. In non-beacon mode, frames are transmitted using unslotted CSMA/CA. Nonetheless, configuring beacons is recommended to facilitate node discovery and manage association and dissociation events. Additionally, for proper network functionality, the source and destination addresses, along with the PAN ID, should be included in the header of an IEEE 802.15.4 frame.

5.2.2 Frame encapsulation and formats

In a LoWPAN network, packets vary based on the transmission requirements. This section outlines the encapsulation format that forms the payload of an IEEE 802.15.4-2011 MAC frame, as we saw in Section 1.1 and depict in Figure 5.1. The 6LoWPAN payload can include either an IPv6 packet or raw data.

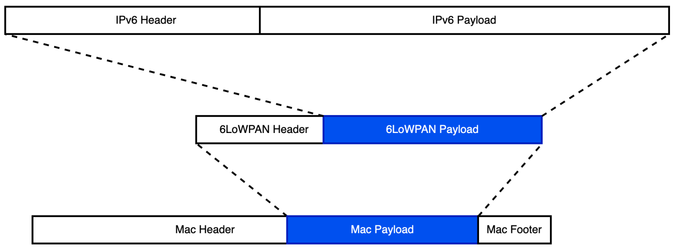


Figure 5.1: Encapsulation of 6LoWPAN datagram in a MAC frame

Each LoWPAN datagram is prefixed with an encapsulated header and may be followed by zero or more header fields. The encapsulation types correspond to specific scenarios and are described on the next page.

When an IPv6 datagram doesn't require compression, fragmentation or meshing, the encapsulation follows this structure:

IPv6 Dispatch	IPv6 Header	Payload

Figure 5.2: Encapsulated IPv6 datagram format

If a compression is necessary, the encapsulation is as follow:

HC1 Dispatch	HC1 Header	Payload

Figure 5.3: Compressed IPv6 datagram format

When both compression and meshing are required, we have the following encapsulation format:

Mesh Type	Mesh Header	HC1 Dispatch	HC1 Header	Payload

Figure 5.4: Compressed and meshed IPv6 datagram format

In cases where compression and fragmentation are need, the encapsulation follows this format:

Frag Type	Frag Header	HC1 Dispatch	HC1 Header	Payload

Figure 5.5: Compressed and fragmented IPv6 datagram

When an IPv6 datagram need compression and requires both mesh addressing and fragmentation to be transmitted, the encapsulation is depicted in the next figure

Mesh Type	Mesh Header	Frag Type	Frag Header	HC1 Dispatch	HC1 Header	Payload

Figure 5.6: Compressed, fragmented and meshed IPv6 datagram format

When compression, mesh addressing and a broadcast header to support mesh broadcast/multicast are necessary, the encapsulation structure is shown below

Mesh Type	Mesh Header	B Dispatch	B Header	HC1 Dispatch	HC1 Header	Payload

Figure 5.7: Compressed and broadcasted IPv6 datagram format

The order in which these encapsulated headers appear for the same packet must follow a precise order

1. The mesh addressing header
2. The broadcast header
3. The fragmentation header

Any datagram that does not contain one of these header encapsulations is considered invalid. This ensures uniform datagram processing.

5.2.3 Dispatch type and header

The dispatch, is a bit pattern that identifies the type of header following the Dispatch Header. The table shown in Figure 5.1 summarises the dispatch bit value associated to their header type and their signification.

Pattern	Header Type	Description
00 xxxxxx	NALP	Invalid LoWPAN encapsulation, discard packet.
01 000001	IPv6	Following header is an uncompressed IPv6 header.
01 000010	LOWPAN_HC1	Compressed header using HC1 compression scheme.
01 010000	LOWPAN_BC0	Use for mesh broadcast/multicast support.
01 111111	ESC	Enables Dispatch values over 127.
10 xxxxxx	MESH	Represents a mesh header.
11 000xxx	FRAG1	Indicates the first fragment header.
11 100xxx	FRAGN	Indicates the nth fragment header.

Table 5.1: Non-Reserved dispatch value bit pattern

Note that other dispatch value exist but are reserved for future use.

5.2.4 Meshing

Meshing refers to the routing and forwarding of packets between devices in a network. In mesh network topology, device can communicate with several others rather than relying solely on a single central point of communication, such as a gateway. The mesh type and header are shown in Figure 5.8.

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 0|V|F|HopsLft| originator address, final address
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5.8: Mesh addressing header

In the mesh header, the 1-bit V and F fields, determine the address format for the originator and final destination, respectively, they are to 0 for an IEEE extended 64-bit address (EUI-64) and 1 for a short 16-bit address. The Hops Left field, is a 4-bit counter that decrements at each node hop. If it reaches 0, the packet is no forwarded. The value `0xF` signals that an 8-bit Deep hops left field follows the header to allow for more than 14 hops. The Originator Address and Final Destination Address fields store the link-layer addresses of the packet's originator and final destination, respectively. V and F fields being independent, we can have a mix of 16 and 64-bit addresses. This is useful to allow for mesh layer broadcast as 802.15.4 broadcast addresses are defined as 16-bit short addresses.

The algorithm which describes how a node should perform the meshing will be explain in detail in the next chapter section 6.4.4, design and implementation.

5.2.5 Fragmentation

Fragmentation is the process in which large data packets are divided into smaller fragments to fit within the MTU of a network protocol. Thus fragmentation is needed only when a payload datagram doesn't fits within a single 802.15.4-2011 frame. All link fragments for a datagram except the last one must be multiple of eight bytes in length.

The first link fragment includes the initial fragment and header as depicted in Figure 5.9, while Figure 5.10 details the header format for the second and all subsequent link fragments.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 0 0 0|    datagram_size    |         datagram_tag          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5.9: First fragment header

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 0 0|    datagram_size    |         datagram_tag          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|datagram_offset|
+-+-+-+-+-+-+-+-+
```

Figure 5.10: Subsequent fragment header

The datagram_size is an 11-bit field that encodes the total size of the IP datagram after IP layer fragmentation but before link-layer fragmentation. This size remains consistent across all fragments of the same IPv6 datagram. The datagram_tag, is a 16-bit field that contains a tag value unique to each IPv6 datagram, shared by all its fragments, and is incremented by the sender for each successive fragmented datagram. Lastly, the datagram_offset is an 8-bit field that indicates the fragment's offset from the beginning of the datagram in 8-byte increments, and is included from the second fragment to the last. The algorithm which describes how a node should handle the reception of fragments will be explain in detail in the next chapter, design and implementation.

5.2.6 Stateless Address Auto-configuration

This subsection explains how to derive an IPv6 interface identifier (IID) from a MAC address, crucial to uniquely identify a network devices on a subnet. As mentioned above, two types of addresses can be assigned to a device, either a 16-bit address or a 64-bit address. In case of a 64-bit address, the interface identifier is formed from this address after having modified the "Universal/Local" `U/L` bit, in accordance with RFC 2464. [35] For a 64-bit MAC address, the IID is formed by modifying the "Universal/Local" (`U/L`) bit as per RFC 2464. [35] The `U/L` bit, the second least significant bit of the first byte, indicates the address's uniqueness. A `U/L` bit of 0 implies a locally administered, non-unique address, while a bit of 1 denotes a universally unique address, ensuring global uniqueness[36]. For 16-bit addresses, we first create a pseudo 48-bit address, to do so, the left most 32 bits are formed by concatenating 16 zeros bits to the 16-bit `PAN ID`. If no `PAN ID` is known, then these 16 bits are replaced by 16 zero bits. Then these 32 bits are concatenated with the 16-bit short address. The final result is a pseudo 48-bit address.

Figure 5.11: Pseudo 48-bit address

The IID is derived from the produced 48-bit address. However, in the resulting Interface Identifier, `U/L` bit must be set to zero.

5.2.7 IPv6 link local address

A link-local address is an IP address valid only within a sub-network to which a host is connected. It is formed by combining a 10-bit prefix `1111111010`, followed by 54 zero bits, and the 64-bit Interface Identifier.

5.2.8 Multicast address mapping

A multicast address identifies a group of network hosts that process datagrams for a specific service. In LoWPAN, multicast addresses are used with meshing and consist of sixteen octets (`DST[1]`. To derive the 16-bit IEEE address from the multicast address, we concatenate the prefix `100`, the last 5 bits of `DST[15]` (bits 3-7), and the 8 bits of `DST[16]` in that order.

5.3 RFC 6282

LOWPAN_HC1 and LOWPAN_HC2 are effective for basic link-local unicast communications but fall short for many IPv6 applications in 6LoWPANs, as they require full in-line IPv6 prefixes and 64-bit Interface Identifiers, even for multicast addresses. RFC 6282 addresses these issues by introducing LOWPAN_IPHC, which enables efficient compression of IPv6 addresses, including Unique Local, Global, and multicast addresses, using shared contexts. In the best case, LOWPAN_IPHC can compress the IPv6 header down to 2 octets for link-local communication and to 7 octets when routing over multiple IP hops. Additionally, it supports the compression of IPv6 Hop Limit values, next headers, and the UDP checksum.

5.3.1 Header compression

To enable effective compression, LOWPAN_IPHC relies on some assumptions about 6LoWPAN communication:

- IP version is always 6.
- Traffic Class and Flowlabel are zero.
- Payload Length is inferred from lower layers.
- Hop Limit is a well-known value set by the source.
- Addresses use the link-local prefix or small set of routable prefixes for the entire 6LoWPAN.
- Addresses are formed with an IID derived from either the 64-bit extended or the 16-bit short IEEE 802.15.4 addresses.

5.3.2 LOWPAN_IPHC encoding format

The base compression format is given below

```
  0                                       1
  0   1   2   3   4   5   6   7   8   9   0   1   2   3   4   5
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 0 | 1 | 1 |   TF    |NH |  HLIM   |CID|SAC|  SAM    | M |DAC|  DAM  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Figure 5.12: LOWPAN_IPHC encoding format

As shown, the encoding can be 2 octets long for the base format, or 3 octets long when additional context encoding is included. The IPv6 header fields that are not fully elided are placed just after the LOWPAN_IPHC header.

5.3.3 Encoding

This section summarises the encoding of the fields in an IPv6 packet.

Traffic Class and Flow Label

TF encodes the Traffic Class and Flow Label fields. Traffic Class consists of a 2-bit Explicit Congestion Notification (ECN) and a 6-bit Differentiated Services Code Point (DSCP). Flow Label spans 20 bits. When a field is set to zero, it is elided, leading to the following TF values:

TF	Description
00	ECN, DSCP + 4-bit zero padding + Flow Label in-line
01	DSCP elided; ECN + 2-bit zero padding + Flow Label in-line
10	Flow Label elided; ECN + DSCP in-line
11	Traffic Class and Flow Label elided

Table 5.2: Traffic class and Flow label compression

Next Header

NH encodes the Next Header field. When set to `0`, all bits of the Next Header are carried in-line. When set to `1`, it indicates that the Next Header is compressed and encoded using LOWPAN_NHC.

NH	Description
0	All bits of the Next Header carried in-line
1	Next Header compressed using LOWPAN_NHC

Table 5.3: Next Header compression

Hop Limit

The HLIM field encodes the Hop Limit as follows, with the assumption that its values are well-known, leading to specific compression methods:

HLIM	Description
00	The Hop Limit bits are carried in-line.
01	When Hop Limit is 1.
10	When Hop Limit is 64.
11	When Hop Limit is 255.

Table 5.4: Hop limit compression

Context Identifier Extension (CID)

The CID selects the context, which refers to pre-defined network prefixes, used for compressing IPv6 addresses. When set to 0, no additional Context Identifier is used, and context 0 is applied for Source Address Compression (SAC) or Destination Address Compression (DAC).

CID	Description
0	No additional 8-bit Context Identifier Extension is used.
1	An additional 8-bit Context Identifier Extension follows the DAM field.

Table 5.5: Context Identifier encoding

Source/Destinaion Address Compression

The Source Address Compression (SAC) and Destination Address Compression (DAC) fields determine whether the compression is stateless or stateful for the source and destination addresses, respectively. Stateless compression relies on shared context or known state, allowing fields like Traffic Class, Flow Label, and Payload Length to be inferred and compressed.

SAC/DAC	Description
0	Stateless compression is used.
1	Stateful, context-based compression is used.

Table 5.6: SAC and DAC encoding

Source Address Mode (SAM)

The Source Address Mode indicates how the source address is compressed depending on the SAC field value

SAC	SAM	Description
0	00	Full 128-bit address in-line
0	01	Last 64 bits in-line; first 64 bits are `fe80::/64`
0	10	Last 16 bits in-line; first 112 bits are `fe80::0000:00ff:fe00`
0	11	Address fully elided; derived from link-local prefix and header
1	00	UNSPECIFIED address, `::`
1	01	Last 64 bits in-line; prefix derived from context
1	10	Last 16 bits in-line; prefix from context, IID from `::ff:fe00:XXXX`
1	11	Address fully elided; derived from shared context

Table 5.7: Source Address Mode encoding

Multicast Compression

M field defines if the destination address is a multiast address.

M	Description
0	DA is not a multicast address.
1	DA is a multicast address.

Table 5.8: Multicast address encoding

Destination Address Mode (DAM)

The Destination Address Mode indicates how the destination address is compressed based on the values of the M and DAC fields.

M	DAC	DAM	Description
0	0	00	Full 128-bit address in-line
0	0	01	Last 64 bits in-line; first 64 bits are `fe80::/64`
0	0	10	Last 16 bits in-line; first 112 bits are `fe80::0000:00ff:fe00`
0	0	11	Address fully elided; derived from link-local prefix and header
0	1	00	Reserved
0	1	01	Last 64 bits in-line; prefix derived from context
0	1	10	Last 16 bits in-line; context provides prefix, IID from `::ff:fe00:XXXX`
0	1	11	Address fully elided; derived from shared context
1	0	00	Full 128-bit address in-line
1	0	01	Last 48 bits in-line; address in form `ffXX::00XX:XXXX:XXXX`
1	0	10	Last 32 bits in-line; address in form `ffXX::00XX:XXXX`
1	0	11	Last 8 bits in-line; address in form `ff02::00XX`
1	1	00	Last 48 bits in-line; Unicast-Prefix-based IPv6 Multicast Address
1	1	01	Reserved
1	1	10	Reserved
1	1	11	Reserved

Table 5.9: Destination Address Mode encoding

5.3.4 Context Identifier extension

The RFC 6282 allows a node to use up to 16 contexts, with source and destination addresses possibly using different contexts. If the CID field in LOWPAN_IPHC encoding is `1`, an extra octet is added after the DAM bits to specify the context pairs for compressing IPv6 source and/or destination addresses. Context 0 is the default.

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
|      SCI      |      DCI      |
+---+---+---+---+---+---+---+---+
```

Figure 5.13: Context Identifier encoding format

The 4-bit SCI and DCI fields identify the prefixes used for compressing the source and destination addresses when using stateful compression.

5.3.5 UDP header compression

The UDP header consists of a `11110` prefix, followed by the 1-bit C and P fields. The C field indicates whether the IPv6 UDP checksum, which is mandatory, is carried in-line or elided if authorized by upper layers with an additional integrity check. The decoder must restore and verify the checksum if it is elided. The P field encodes port values, where ports in the range `0xf0b0` to `0xf0bf` can be compressed to 4 bits. This range should be avoided for dynamic ports to prevent payload misinterpretation. The C and P fields compression are given in Table 5.10 and 5.11

C	Description
`0`	Full 16-bit checksum carried in-line.
`1`	Checksum elided and recomputed at the 6LoWPAN endpoint.

Table 5.10: UDP Checksum encoding

P	Description
`00`	Full 16 bits for both SP and DP carried in-line.
`01`	Full SP, last 8 bits of DP in-line; first 8 bits of DP (`0xf0`) elided.
`10`	Full DP, last 8 bits of SP in-line; first 8 bits of SP (`0xf0`) elided.
`11`	First 12 bits of SP and DP are elided; last 4 bits for each carried in-line.

Table 5.11: UDP Ports encoding

Chapter 6

Design and implementation

In this chapter, we will discuss the choices that have been made to implement the features of the 6LoWPAN standard. In particular, we will look at the architecture of the code, the nature and role of each component, and the different algorithms used, including concrete code examples.

6.1 Code architecture

The implementation developed in this work is based on the architecture depicted in Figure 6.1. Note that this architecture only includes the essential modules, the remaining components consist of the various header files, test files, and the modules specific to the IEEE 802.15.4 layer discussed in the previous work section.

6.1.1 Code architecture overview

The code architecture comprises five modules, four of which; `lowpan api`, `lowpan core`, `routing table`, and `lowpan ipv6`; were developed as part of this thesis. The `Ieee802154` module, inherited from previous work, defines the IEEE 802.15.4 MAC layer and serves primarily as an API for the 6LoWPAN layer, enabling it to transmit and receive data frames. Below is an overview of each module and their interactions:

- **`lowpan api`**: This module provides an API for the 6LoWPAN layer, enabling IPv6 packets to be transmitted to the IEEE layer through compression, fragmentation, and meshing mechanisms. It also handles the reception of frames from the IEEE layer.
- **`lowpan core`**: This module serve as a core engine for the `lowpan api`. It implements the key mechanisms of 6LoWPAN and includes essential functions

such as `compressIpv6Header`, `triggerFragmentation`, and `decodeIpv6Pckt`.

- **`routing table`**: Initialized along with the `lowpan api`, this module manages the routing table, providing an API for adding, deleting, updating, and finding routes between network nodes. It is also utilized by the `lowpan core` during meshing operations.
- **`lowpan ipv6`**: This module contains functions to create IPv6 and UDP packets.

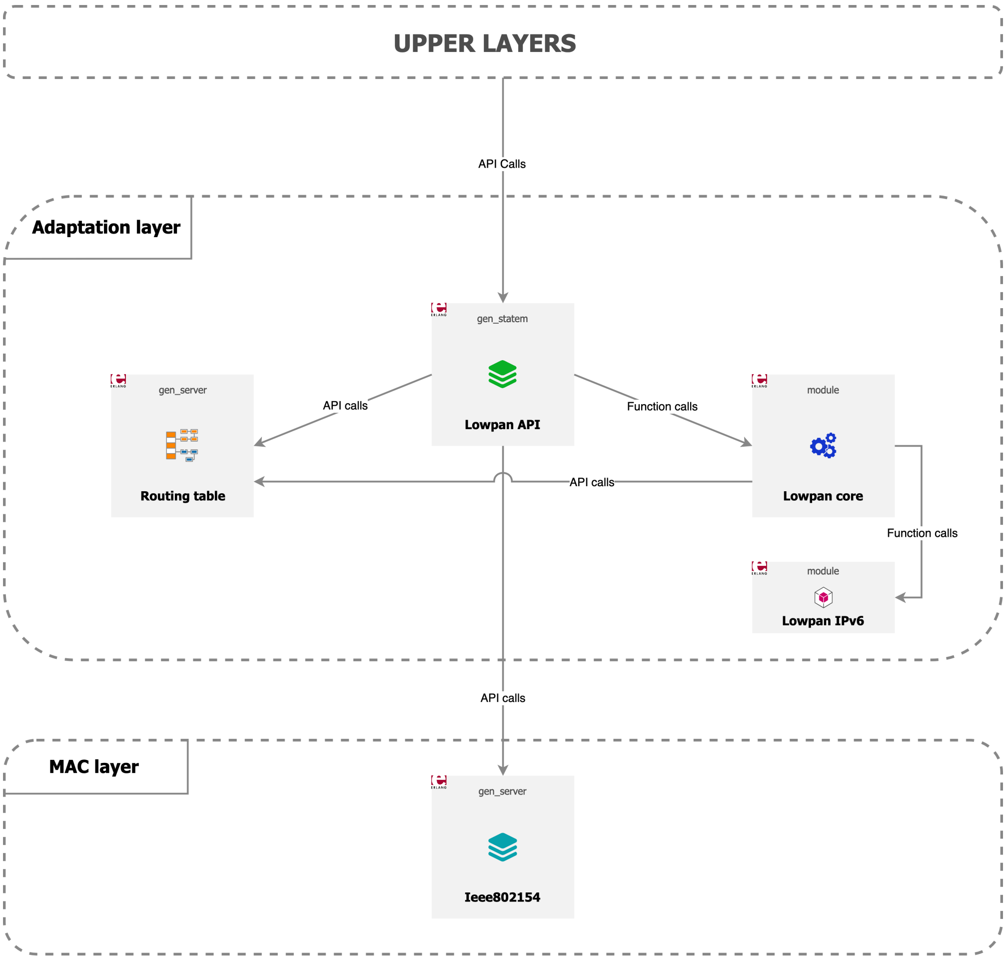


Figure 6.1: Code architecture

6.2 Lowpan API

The `lowpan api` is designed as a finite state machine, providing a clear and structured way to manage the various transitions that occur when API requests are processed by the 6LoWPAN layer. This state machine approach ensures that the different states and transitions within the API are explicitly defined, making the behavior of the system more predictable and easier to understand.

Erlang facilitates the implementation of state machines through a mechanism called Behaviours. Behaviours allow code for a process to be split into a generic component (the behaviour module) and a specific component (the callback module). These behaviours encapsulate common design patterns and provide a standardized structure for modules, enabling the implementation of servers, finite state machines, event handlers, and supervisor processes [37]. The 6LoWPAN API's behaviour is modeled using Erlang's `gen_statem`, a finite state machine module. In the next subsection, we will explore the definition and features of `gen_statem` in detail.

6.2.1 Generic state machine

The `gen_statem` behavior is designed to implement event-driven state machine. In traditional automate theory, state transitions are typically triggered by inputs, and the output is a function of the input and the current state. However, in an event-driven state machine, the input is considered as an event that triggers state transitions and actions. The state machine can be described with the following relation

$$\text{State}(S) \times \text{Event}(E) \rightarrow \text{Action}(A), \text{State}(S')$$

This simply means that if the machine is in State S and an event E occurs, it performs action an A and transitions to the state S'. It is important to point out that S' can be the same as S and actions A can be empty. [38]

`gen_statem` offers several features that simplify the management of complex state machines. Let's examine the ones that were particularly useful during implementation.

State callback

When an event occurs the `gen_statem` engine calls a function in the callback module with the event, current state, and server data. This function then processes the event, performs the required actions, returns a new state and update server data. `gen_statem` supports two callback modes

1. **`state_functions`**: In this mode, each state has its own callback function.

```
packet_tx(EventType, EventContent, Data) ->
    % Handle events in the 'packet_tx' state
```

2. **handle_event_function**: In this mode, all events are handled by a single callback function.

```
handle_event(EventType, EventContent, State, Data) ->
    % Handle events for all states
```

Event types

Events are categorized into different types, which determine how they are handled

- **call**: Used to send synchronous messages.
- **cast**: Used to send asynchronous messages.
- **internal**: Used to generate internal event within the state machine.

```
idle(cast, {frame_rx, From}, _Data)-> % Handle asychronous call
```

Transition actions

Transition actions are operations that the state machine performs as it transitions from one state to another. These actions are returned by the state callback function. Some common actions include

- **next_event**: Used to generate the next event to handle internally.
- **reply**: Used to send a reply to a caller process.

```
{next_state, idle, NewData [{reply, From, ok}]} % Go to idle state and reply ok
```

Inserted events

Inserted events allow the state machine to trigger new events that should be handled after the current transition is complete. They are useful for internal processing.

```
{next_state, NextStateName, NewData, [{next_event, internal, {someState, Data}}]}
```

Start and stop the gen_statem

To start the `gen_statem` we can use the `start_link` function. This function call will spawn a new process that will run the state machine and link it to the supervisor process managing the state machine. In Erlang, a supervisor is a process that oversees other processes, known as child processes. It is responsible for starting, stopping, and monitoring these processes to ensure that they are running correctly. If a child process fails, the supervisor can restart the process, to maintain the system's robustness and fault tolerance.

```
start_link(Params) ->
    gen_statem:start_link({local, ?NAME}, ?MODULE, Params, []).
```

`{local, ?NAME }` locally registers the process with a given name. `?MODULE` is the module where the callback functions are defined.

To stop the state machine process, we use the `stop` function, we can stop the .

```
stop() ->
    gen_statem:stop(?MODULE).
```

6.2.2 Lowpan API state machine diagram

This sub-section details the operation of the Lowpan API state machine. The state diagram related to the lowpan API is illustrated in Figure 6.2.

Initialisation phase

When the state machine process starts, the `init` function is called. In this function, the state machine process is started, then, using the attributes passed as parameters, the MAC address of the node is retrieved and stored. This will later be used to check if a received packet has reached its destination or not. After this, the routing table is launched, along with the IEEE 802.15.4 stack. Finally, the state machine transitions to the idle state, the default state of the machine.

Transmission phase

In the idle state, when a transmission request is received, the state machine first validates the packet or datagram. For IPv6 packets, it checks the destination and source addresses: the source address cannot be a multicast address, as these are intended for groups of nodes, and the destination address cannot be unspecified, like the unspecified IPv6 address. For 6LoWPAN datagrams, the state machine checks the datagram validity; if not, it returns the error `error_nalp`.

The `lowpan_api` supports three types of transmissions. The first allows to send an IPv6 packet, the second a datagram (an IPv6 packet that should not be compressed, fragmented, or meshed), and third, a basic frame. Upon receiving a transmission request, the appropriate event will be triggered, either `frame_tx` to send a frame, `packet_tx` to send a packet, or `datagram_tx` to send a datagram. These events cause the state machine to transition to their corresponding state; `Tx frame`, `Tx packet`, or `Tx datagram`; using asynchronous calls. It is within these states that the actual data transmission will occur. If the transmission has been successful, an `ok` message is sent to the calling process, closing the transmission phase and returning the machine to the idle state. If a failure occurred during the transmission, an error message will be sent back to the calling process.

Reception phase

In the idle state, when a packet reception request is received, the machine transitions to the `Rx frame` state via an asynchronous call. In this state, it waits to receive a frame from the IEEE MAC layer. If no message is received within 60 seconds, the maximum time for packet reassembly as defined in RFC 4944, a reassembly timeout message is sent to the calling process, and the machine returns to the idle state.

If a frame is received, the destination address is compared with the current node's address. There are two possibilities, either the frame has reached its final destination, or it has not. If the final destination is not reached, the `start_forward` event is triggered, and the machine transitions to the `forward` state. In this state, the remaining hop count is verified. If the hop count is greater than zero, the packet is forwarded to the next node, and the machine returns to the `Rx frame` state. If the hop count is zero or less, the packet is discarded, and the machine returns to the `Rx frame` state. This will result in a reassembly timeout since the packet will never be fully received at the destination, indicating that the hop count was insufficient. When the received packet reaches its destination, it is checked to determine whether it is a fragment or a complete packet. If it is a complete packet, the `complete` event is triggered, the packet is decoded and forwarded to the calling process, and the machine transitions back to the `idle` state. If the packet is fragmented, the `start_collect` event is triggered, causing the machine to switch to the `collect` state. In this state, fragments associated with the originator node address are collected until all fragments are received. After storing each fragment, the machine switches back to the `Rx frame` state. If all fragments are not collected before the timeout, the associated entry in the storage table is deleted, an error message is sent, and the machine returns to the idle state. Once all fragments are received, the `complete` event is triggered, transitioning the machine to the `reassemble` state. Here, the fragments are assembled into a complete packet, which is then decoded and sent back to the calling process. After this final step, the machine transitions to the `idle` state, completing the reception phase.

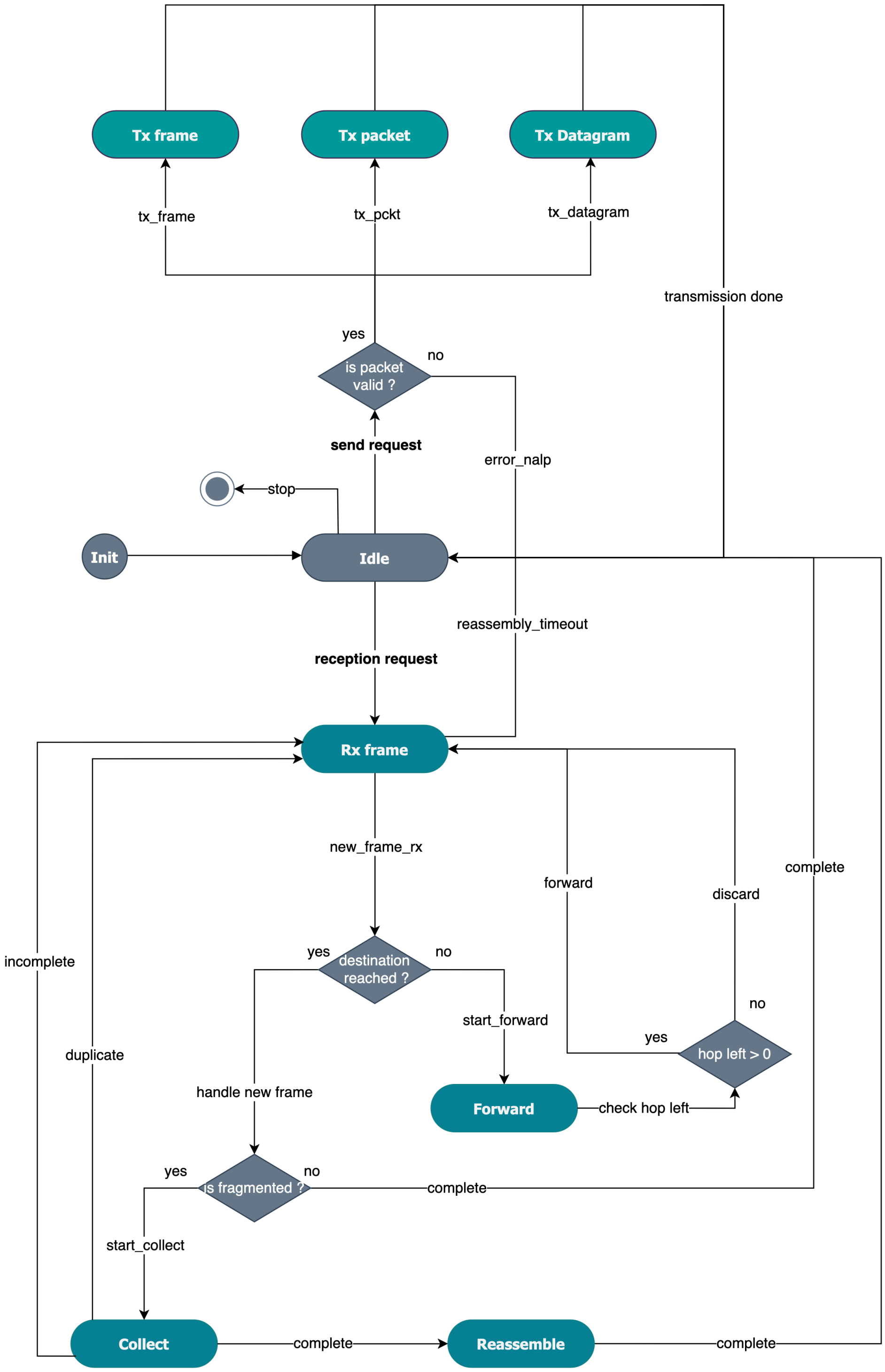


Figure 6.2: Lowpan API state diagram

6.3 Lowpan API

In this section, we'll take a detailed look at the algorithms describing the transmission and reception phases of the state machine.

6.3.1 Packet transmission algorithm

The packet transmission algorithm described here, focus on the transmission logic of an IPv6 packet, as the higher layers will primarily use this transmission mode to benefit from the advantages of 6LoWPAN.

Algorithm 1 Tranmission of an IPv6 packet

Implements: LowpanAPI, **instance** *lowpan_api*
Uses: LowpanCore, **instance** *lowpan_core*; RoutingTable, **instance** *routing_table*;

```
upon event ⟨lowpan_api, tx_pckt | IPv6Pckt, PcktInfo, ExtendedHopsLeft, From⟩ do
    {DestMacAddr, SenderMacAddr} ← lowpan_api:getEUI64MacAddr
    MacAddrs ← {DestMacAddr, SenderMacAddr}
    {RouteExist, MeshedHdrBin, MH} ← lowpan_core:getNextHop(PcktInfo, MacAddrs)
    CompressedHeader ← lowpan_core:compressIPv6Header(IPv6Pckt,RouteExist)
    CompressedPckt ← buildPacket(CompressedHeader, PcktInfo)
    {FragReq, Fragments} ← lowpan_core:triggerFragmentation(CompressedPckt, RouteExist, Tag)
    if FragReq == TRUE then
        Response ← sendFragments(Fragments, MeshedHdrBin, MH)
        reply Response to From
    else if FragReq == FALSE then
        Response ← sendFragment(Fragments, MeshedHdrBin, MH)
        reply Response to From
    else
        reply error_frag_size to From
```

Explanation: Upon receiving the `txPacket` event, the next hop is first determined based on the IPv6 packet information and the node's MAC addresses. The IPv6 header is then compressed, followed by a check to determine if fragmentation is necessary. If fragmentation is required, the `triggerFragmentation` function generates a list of fragments, which are sent to the next hop. If fragmentation is not needed, the entire packet is transmitted. In both cases, a response is sent back to the calling process. Finally, if the packet's payload size exceeds the allowable limit, an error message is returned. The fragmentation and compression algorithm will be introduce in Section 6.4.

The API function to send an IPv6 packet is `sendPacket` and takes an IPv6 packet in binary form as parameter.

6.3.2 Frame reception algorithm

Algorithm 2 Handle new frame

```
Uses: LowpanCore, instance lowpan_core; LowpanAPI, instance lowpan_api,
IEEE802154; instance ieee802154, RoutingTable; instance routing_table
upon event ⟨InputCallback, new_frame | Frame⟩ do
  CurrNodeMacAdd ← lowpan_api:getCurrentNodeAddr()
  {FC, MH, Datagram} ← Frame
  {IsMeshedPckt, FinalDstMacAdd, MeshPckInfo} ← lowpan_core:containsMeshHeader(Datagram)
  if FinalDstMacAdd == CurrNodeMacAdd then
      if Datagram == complete then
          ReassembledPacket ← lowpan_core:decodeIpv6Pckt(Datagram)
          reply ReassembledPacket to From
      else
          StoringResult ← lowpan_core:StoreFragment(Datagram)
          if StoringResult == complete then
              {StorageCode, DatagramMap} ← StoringResult
              ReassembledPacket ← lowpan_api:reassemble(DatagramMap, Key)
              DecodedPacket ← lowpan_core:decodeIpv6Pckt(reassembledPacket)
              reply DecodedPacket to From
          else if StoringResult ≠ complete ∧ timeout then
              deleteEntry(DatagramMap, Key)
              reply reassembly_timeout to From
          end if
      end if
  else
      if IsMeshedPckt then
          if HopLft == 0 then
              reply dtg_discarded to From
          else
              NewMeshHeader ← updateMeshHeader(MeshPcktInfo)
              NewDatagram ← createNewDatagram(NewMeshHeader)
          end if
      else
          NewDatagram ← createNewMeshDatagram(MeshPcktInfo)
      end if
      NextHopAddr ← routing_table:getRoute(FinalDstMacAdd)
      Frame ← {FC, MH, NewDatagram}
      ieee802154:transmission(Frame)
  end if=0
```

Explanation: When a new frame is received, the current node's MAC address and key fields, like the Mesh Header (MH) and datagram, are extracted. If the final destination MAC address matches the current node's address and the datagram isn't fragmented, the packet is decoded and sent. If fragmented, the fragment is stored and once all fragments arrive, reassembled, then decoded and sent. If reassembly times out, the entry is deleted, and a timeout error is returned to the calling process. For meshed datagrams, the hop left is checked. If zero, the packet is discarded, otherwise, the mesh header is updated and a new datagram is created. Non-meshed datagrams are converted to mesh datagrams. Finally, the next hop address is determined from the routing table, and the datagram is forwarded. The API function to receive a datagram packet is `frameReception`.

6.4 6LoWPAN core

This section explains how the features described in RFC 4944 and RFC 6282 have been implemented, with associated algorithms and concrete code examples.

6.4.1 Header compression

To compress an IPv6 header, key fields such as traffic flow, next header, and source/destination addresses are extracted from the IPv6 packet. These fields are then encoded following the compression scheme as defined in RFC 6282 (Section 5.3.3). If the next header is UDP, the UDP header is also compressed and appended to the IPv6 header. Otherwise, the compressed IPv6 header is returned directly. The `compressIpv6Header` function handles this process, taking a binary IPv6 packet and a boolean flag to indicate if the packet is meshed, allowing address elision since they are included in the mesh header. The function returns the compressed header in binary format. An example of compression is given in the following code. It shows the encoding of the dam field, when `M = 1 and DAC = 1`. In this particular case, the last 48 bits of the destination address are carried in-line.

```
encodeDam(_CID, 1, 1, DstAdd, CarrInlineMap, CarrInlineList, _RouteExist) ->
    DestAddBits = <<DstAdd:128>>,
    <<_:80, Last48Bits:48>> = DestAddBits,
    case DestAddBits of
        <<16#FF, _:112>> ->
            Bin = <<Last48Bits:48>>,
            L = [Bin],
            UpdatedList = [CarrInlineList, L],
            UpdatedMap = CarrInlineMap#{"DAM" => Bin},
            {2#00, UpdatedMap, UpdatedList}
    end.
```

The header compression code is given in the appendix.

6.4.2 Next header compression

The Next header compression is perform using the the `compressUdpHeader` function. To compress a UDP header, if present, fields such as source port, destination port, and checksum are extracted from the Ipv6 packet and encoded as defined in RFC 6282 (Table 5.10, 5.11). The resulting compression is then appended to the compressed header to form the complete compressed UDP header, which is returned. An example of compression is given in the following code. It shows the encoding of Source and destination ports where the first 12 bits of both source port and destination port are `0xf0b` and elided and the remaining 4 bits for each are carried in-line.

```
encodeUdpPorts(SrcPort, DstPort, CarriedInline) ->
```

```
    case {<<SrcPort:16>>, <<DstPort:16>>} of
        {<<?0xf0b:12, Last4S_Bits:4>>, <<?0xf0b:12, Last4D_Bits:4>>} ->
            ToCarr = <<Last4S_Bits:4, Last4D_Bits:4>>,
            L = [ToCarr],
            CarriedInlineList = CarriedInline ++ L,
            P = 2#11,
            {P, CarriedInlineList}
    end.
```

The code of the UDP next header compression can be found in the appendix.

6.4.3 Fragmentation

The fragmentation process is performed using the `triggerFragmentation` function whose algorithm is described below.

Algorithm 3 Fragmentation

```
Uses: lowpanCore, instance lowpan_core
Input: CompPckt, DatagramTag, RouteExist
Output: {isFragmented, FragmentList}
  if byte_size(CompPckt) ≤ MAX_DTG_SIZE then
      if byte_size(CompPckt) > MAX_FRAME_SIZE then
          Fragments ← []
          PcktSize ← byte_size(CompPckt)
          Offset ← 0
          if RouteExist then
              MaxFragSize ← MAX_FRAG_SIZE_MESH
          else
              MaxFragSize ← MAX_FRAG_SIZE_NoMESH
          end if
          while CompPckt ≠ empty do
              FragmentSize ← min(PcktSize, MaxFragSize)
              FragPayload ← CompPckt[0:FragmentSize]
              if Offset == 0 then
                  Header ← buildFirstFragHeader(DatagramTag, PcktSize, Offset)
              else
                  Header ← buildFragHeader(DatagramTag, PcktSize, Offset)
              end if
              Fragments ← Fragments + [{Header, FragPayload}]
              CompPckt ← CompPckt[FragmentSize:]
              Offset ← Offset + 1
          end while
          return {true, Fragments}
      else
          return {false, CompPckt}
      end if
  else
      return {size_err, error_frag_size}
  end if
```

Explanations: When a compressed packet (`CompPckt`) is received, a first check is done to verify if its size is within the maximum allowable datagram size, defined as `MAX_DTG_SIZE`, which corresponds to an 11-bit encoding limit. If the packet exceeds the maximum frame size (`MAX_FRAME_SIZE = 80 bytes`), the fragmentation is triggered. This maximum frame size is calculated considering the worst-case scenario for the MAC layer, where additional fields, particularly those related to security, are included in the IEEE 802.15.4-2011 MAC header. If meshing is needed, the maximum fragment size is set to `MAX_FRAG_SIZE_MESH` (58 bytes), considering space taken by the mesh and fragment headers. Without meshing, the maximum fragment size is `MAX_FRAG_SIZE_NoMESH` (75 bytes), as the absence of a mesh header allows more room for the payload. The packet is then divided into appropriately sized fragments, with each fragment receiving a corresponding header. The first fragment uses a `Frag1` header, while subsequent fragments use `FragN` headers. These fragments are collected into a list and returned. If fragmentation isn't needed, the original packet is returned ans if the packet size exceeds `MAX_DTG_SIZE`, an error is returned.

6.4.4 Meshing

The meshing process is performed using the `getNextHop` function whose algorithm is described below.

Algorithm 4 Meshing

```
Uses: LowpanCore, instance lowpan_core; RoutingTable, instance routing_table,
  Input: CurrNodeMacAdd, SenderMacAdd, DestMacAddress, DestAddress, SeqNum, Hopsleft_extended
  Output: {isMeshed, Header, MHdr}
  if DestAddressPrefix == Multicast then
      MulticastAddr ← lowpan_core:generateMulticastAddr(DestAddress)
      MulticastEU64 ← lowpan_core:generateEUI64MacAddr(MulticastAddr)
      MH ← createMacHeader(CurrNodeMacAdd, MulticastEU64)
      BroadcastHeader ← lowpan_core:createBroadcastHeader(SeqNum)
      MeshHdrBin ← lowpan_core:createNewMeshHeader(SenderMacAdd, DestMacAddress, Hopsleft_extended)
      Header ← concat(MeshHdrBin, BroadcastHeader)
      return {false, Header, MH}
  else
      NextHopMacAddr ← routing_table:getRoute(DestMacAddress)
      if NextHopMacAddr ≠ DestMacAddress then
          MH ← createMacHeader(CurrNodeMacAdd, NextHopMacAddr)
          MeshHdrBin ← lowpan_core:createNewMeshHeader(SenderMacAdd,DestMacAddress,Hopsleft_extended)
          return {true, MeshHdrBin, MacHdr}
      else if NextHopMacAddr == DestMacAddress then
          MHdr ← {src_addr = CurrNodeMacAdd, dest_addr = DestMacAddress}
          return {false, emptyBin, MHdr}
      else
          return {false, emptyBin, undefined, undefined}
      end if
  end if
```

Explanations: When meshing is required, the destination address prefix is checked to determine if the received address is a multicast address or not. If it is multicast, a corresponding multicast MAC address is generated, and the appropriate headers, including a broadcast and mesh header, are created, then the mesh header is returned. For unicast addresses, the next hop MAC address is retrieved from the routing table. If intermediate routing is necessary, a mesh header is created and returned with `isMeshed` set to `true`. If the next hop is the destination, no meshing is necessary so `isMeshed` set to `false` and only the MAC header is returned. If no valid route is found, an undefined error is returned.

6.4.5 Stateless address generation

The stateless address generation is done through the `generateEUI64MacAddr` function and follow the steps described in section 5.2.6. As a reminder, for 16-bit addresses, the address is expanded to 48 bits by concatenating the `PanID` and 16-bit zeros to the MAC address. The `U/L` bit is then adjusted, and `0xFFFE` is inserted between the first 24 bits and the last 24 bits to form the `EUI-64`. For 64-bit addresses, the `U/L` bit is simply flip. The code is given below.

```
-spec generateEUI64MacAddr(binary()) -> binary().
generateEUI64MacAddr(MacAddress) when byte_size(MacAddress) == ?SHORT_ADDR_LEN ->
    PanID = <<16#FFFF:16>>,
    Extended48Bit = <<PanID/binary, 0:16, MacAddress/binary>>,
    <<A:8, Rest:40>> = Extended48Bit,
    ULBSetup = A band 16#FD,
    <<First:16, Last:24>> = <<Rest:40>>,
    EUI64 = <<ULBSetup:8, First:16, 16#FF:8, 16#FE:8, Last:24>>,
    EUI64;
generateEUI64MacAddr(MacAddress) when byte_size(MacAddress) == ?EXTENDED_ADDR_LEN
    ->
    <<A:8, Rest:56>> = MacAddress,
    NewA = A bxor 2,
    <<NewA:8, Rest:56>>.
```

6.4.6 Reassembly

The reassembly process is performed in the `reassemble` function. The logic is quite simple, the fragments of a datagram are first retrieved and sorted by their offset. These fragments are then combined into the reassembled datagram, which is returned as the final output.

6.4.7 Header decoding

Decoding is handled by the `decodeIpv6Pckt` function, which applies the reverse process of the compression. It reconstructs the original IPv6 packet by interpreting the compressed header fields and in-line data, utilizing the corresponding decoding functions. The following code example shows the decoding of the TF field when `TF = 10` to retrieve the TrafficClass and FlowLabel values.

```
decodeTf(TF, CarriedInline) ->
    case TF of
        2#10 -> <<ECN:2, DSCP:6, Rest/bitstring>> = CarriedInline,
```

```
            {{DSCP, ECN}, 0, Rest}
    end.
```

6.5 Routing table

The routing table, implemented using the Erlang `gen_server` behavior, manages route addition, deletion, updating, and retrieval. The initialization of the routing table is generally done via the neighbor discovery procedure however, the design of a complete routing algorithm was beyond the scope of this work. Instead, the table is initialized with a pre-filled map at start-up. This map links destination nodes to next-hop 64-bit MAC addresses. A more complex design could support multiple routes per destination, but this implementation simplifies this process by returning the first route found. Let's take a brief look at how `gen_server` works in Erlang.

6.5.1 gen_server

In Erlang the `gen_server` is designed to simplify the implementation of the client-server model, where a central server manages shared resources and multiple clients interact with it. It provides a framework to manage the lifecycle of a server process, handle client requests, and maintain the server's state. [39] Similarly to the `gen_statem`, the gen server can be started using the `start_link` function.

```
gen_server:start_link({local, ?MODULE}, ?MODULE, RoutingTable, []).
```

It also supports synchronous and asynchronous request. The following code example shows a synchronous request management for adding a new route in the routing table.

```
handle_call({add_route, DestAddr, NextHAddr}, _From, RoutingTable) ->
    NewTable = maps:put(DestAddr, NextHAddr, RoutingTable),
    {reply, ok, NewTable};
```

Finally it can be stop using the `stop` function.

```
stop() ->
    gen_server:stop(?MODULE).
```

Chapter 7

Tests and results

In this section, we will discuss the tests that were performed to validate the the implementation. They are divided into two groups, software tests and hardware tests on the GRiSP 2 board.

7.1 Hardware constraints

Due to the unavailability of GRiSP 2 boards for a large part of this thesis, I relied on software simulations using mockups provided by Stritzinger GmbH engineer Gwendal Laurent, simulating the physical layer. It was only toward the end of the year that I could validate my work on the actual hardware.

7.2 Default field values

Several default values were defined for generating IPv6 and UDP packets. In certain test cases, these values were adjusted to ensure relevance. The table below outlines these values.

Field	**value**
Traffic class	0
Flow label	0
Payload length	54
Next header	12
Hop limit	64
Source address	`FE:80:00:00:00:00:00:00:C8:FE:DE:CA:00:00:00:01`
Destination address	`FE:80:00:00:00:00:00:00:C8:FE:DE:CA:00:00:00:02`
Payload	"Hello world this is an ipv6 packet for testing purpose"
Datagram tag	`0x007c`
Sequence number	`3`
UDP source port	1025
UDP destination port	61617
UDP checksum	`0xf88c`

Table 7.1: Default field values

7.3 Software tests

The Erlang Common Test framework was utilized for software testing. This tool simplifies the creation and automated execution of test cases across various target systems, allowing tests to be run individually or grouped together. [40] Two types of software tests were conducted: functional tests to validate the correctness of functions in the `lowpancore` module and simulation tests to model various communication scenarios.

7.3.1 Function validation tests

Among the tests performed, here is a description of a few key examples.

Test Name	**Test Description**
pkt_encapsulation_test	Tests the encapsulation of an IPv6 packet by verifying that it is correctly wrapped with the appropriate 6LoWPAN header.
iphc_pckt_64bit_addr	Tests the IPv6 header compression function when using 64-bit addresses.
msh_pckt	Validates the construction of a 6LoWPAN mesh header.
link_local_addr_pckt_comp	Ensures the compression of IPv6 headers using link-local addresses is performed correctly, reducing the header size as expected.
multicast_addr_pckt_comp	Validates that IPv6 headers containing multicast addresses are compressed properly.
global_context_pckt_comp1	Tests the compression of IPv6 headers using global context address.
udp_nh_pckt_comp	Checks the compression of IPv6 headers where UDP is the next header, ensuring correct compression and inclusion of UDP-specific fields.
compress_header_example1_test	Uses the online example to ensure the correctness of the compression. [41]
fragmentation_test	Tests the fragmentation of an IPv6 packet into smaller 6LoWPAN fragments, then reassembles them to ensure the fragments correctly reconstruct the original packet.
reassemble_full_ipv6_pckt_test	Tests the reassembly process for a fully fragmented IPv6 packet, ensuring all fragments are stored, processed, and reconstructed into the original packet.
extended_EUI64_from_64mac	Verifies the correct handling of converting a 64-bit MAC address to a standardized EUI64 format, maintaining consistency with the 6LoWPAN requirements.
extended_EUI64_from_16mac	Ensures that a 16-bit short MAC address is correctly converted into a EUI64 address.
multicast_addr_validity	Tests the validity of generated multicast addresses, ensuring that the multicast address conversion is correct.

Table 7.2: Function validation tests

As shown in Figure 7.1, all 24 tests performed were successfully passed.

1	lowpan_test_SUITE		pkt_encapsulation_test	< >	0.000s	Ok	
2	lowpan_test_SUITE		datagram_info_test	< >	0.000s	Ok	
3	lowpan_test_SUITE		reassemble_fragments_list_test	< >	0.000s	Ok	
4	lowpan_test_SUITE		reassemble_single_fragments_test	< >	0.000s	Ok	
5	lowpan_test_SUITE		reassemble_full_ipv6_pckt_test	< >	0.002s	Ok	
6	lowpan_test_SUITE		compress_header_example1_test	< >	0.000s	Ok	
7	lowpan_test_SUITE		compress_header_example2_test	< >	0.000s	Ok	
8	lowpan_test_SUITE		link_local_addr_pckt_comp	< >	0.000s	Ok	
9	lowpan_test_SUITE		multicast_addr_pckt_comp	< >	0.000s	Ok	
10	lowpan_test_SUITE		global_context_pckt_comp1	< >	0.000s	Ok	
11	lowpan_test_SUITE		udp_nh_pckt_comp	< >	0.000s	Ok	
12	lowpan_test_SUITE		tcp_nh_pckt_comp	< >	0.000s	Ok	
13	lowpan_test_SUITE		icmp_nh_pckt_comp	< >	0.000s	Ok	
14	lowpan_test_SUITE		unc_ipv6	< >	0.000s	Ok	
15	lowpan_test_SUITE		iphc_pckt_64bit_addr	< >	0.000s	Ok	
16	lowpan_test_SUITE		iphc_pckt_16bit_addr	< >	0.000s	Ok	
17	lowpan_test_SUITE		msh_pckt	< >	0.000s	Ok	
18	lowpan_test_SUITE		extended_EUI64_from_64mac	< >	0.000s	Ok	
19	lowpan_test_SUITE		extended_EUI64_from_48mac	< >	0.000s	Ok	
20	lowpan_test_SUITE		extended_EUI64_from_16mac	< >	0.000s	Ok	
21	lowpan_test_SUITE		check_tag_unicity	< >	0.000s	Ok	
22	lowpan_test_SUITE		link_local_from_16mac	< >	0.000s	Ok	
23	lowpan_test_SUITE		multicast_addr_validity	< >	0.000s	Ok	
24	lowpan_test_SUITE		broadcast_pckt	< >	0.000s	Ok	
	common_test		end_per_suite	< >	0.000s	Ok	
	TOTAL				**0.006s**	**Ok**	**24 Ok, 0 Failed of 24** **Elapsed Time: 0.693s**

Figure 7.1: Functional testing results

7.3.2 Simulation testing

In this section, we will review the scenarios evaluated during the simulation phase. Given the document size limit, we will discuss 5 of the 18 scenarios but note that the complete code for all test scenarios is available in the appendix.

7.3.2.1 Transmission/reception simulation workflow

Figure 7.2 illustrates the communication process between two nodes using the simulation. When Node 1 transmits data, the process starts with API calls to the 6LoWPAN layer, which triggers the IEEE 802.15.4 MAC layer to send the data using the physical mock-up. Upon reception, Node 2 follows the reverse sequence, the physical mock-up captures the data, passes it to the MAC layer, and the 6LoWPAN layer processes it through a callback function.

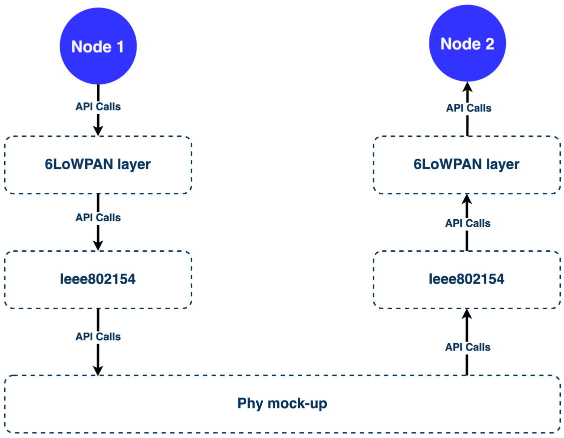


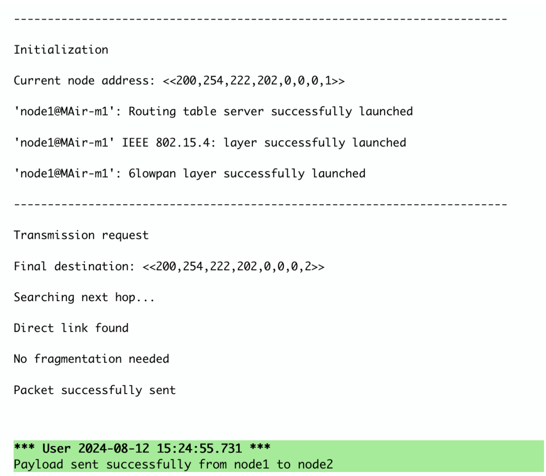
Figure 7.2: Software test workflow

7.3.2.2 Basic packet transmission

Description Transmission of an IPv6 packet that doesn't need to be fragmented. Node 1 is the sender and node 2 the receiver.

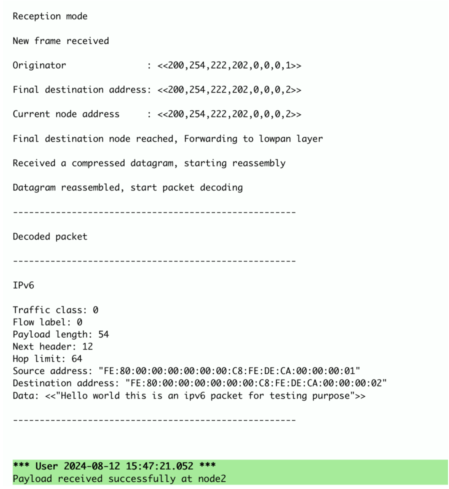
Expected results The packet should correctly be transmitted by node 1, received and decoded at node 2.

Simulation output



```
------------------------------------------------------------------------

Initialization

Current node address: <<200,254,222,202,0,0,0,1>>

'node1@MAir-m1': Routing table server successfully launched

'node1@MAir-m1' IEEE 802.15.4: layer successfully launched

'node1@MAir-m1': 6lowpan layer successfully launched

------------------------------------------------------------------------

Transmission request

Final destination: <<200,254,222,202,0,0,0,2>>

Searching next hop...

Direct link found

No fragmentation needed

Packet successfully sent


*** User 2024-08-12 15:24:55.731 ***
Payload sent successfully from node1 to node2
```

Figure 7.3: Simple transmission Sender



```
Reception mode

New frame received

Originator              : <<200,254,222,202,0,0,0,1>>

Final destination address: <<200,254,222,202,0,0,0,2>>

Current node address     : <<200,254,222,202,0,0,0,2>>

Final destination node reached, Forwarding to lowpan layer

Received a compressed datagram, starting reassembly

Datagram reassembled, start packet decoding

-----------------------------------------------------

Decoded packet

-----------------------------------------------------

IPv6

Traffic class: 0
Flow label: 0
Payload length: 54
Next header: 12
Hop limit: 64
Source address: "FE:80:00:00:00:00:00:00:C8:FE:DE:CA:00:00:00:01"
Destination address: "FE:80:00:00:00:00:00:00:C8:FE:DE:CA:00:00:00:02"
Data: <<"Hello world this is an ipv6 packet for testing purpose">>

-----------------------------------------------------


*** User 2024-08-12 15:47:21.052 ***
Payload received successfully at node2
```

Figure 7.4: Simple transmission Receiver

Observations As the simulation output shows, after the initialization phase, the sender sends a transmission request to the 6LoWPAN layer. Once the request has been received, a search is performed in the routing table to send the packet to the next hop. In this case, a direct link has been found, which means that the 2 nodes are in the same range and can communicate directly. The last message tells us that the transmission was successful. On the receiver side, after the initialisation phase, it goes into receive mode. When the frame is received, a check is performed to determine whether the frame has reached its intended destination. This is the case here, the packet is then decoded, and the decoding results corresponds to the packet initially transmitted. This confirms the expected behavior.

7.3.2.3 Big payload transmission

Description Transmission of an IPv6 packet containing a 290-byte payload. Node 1 is the sender and node 2 the receiver.

Expected results Given the payload size, and the compressed header size (19 bytes), the packet should be fragmented in 5 fragments in order to be send to node 2. At node2, all fragments should be stored, reassemble once the transmission is over then decoded.

Simulation output

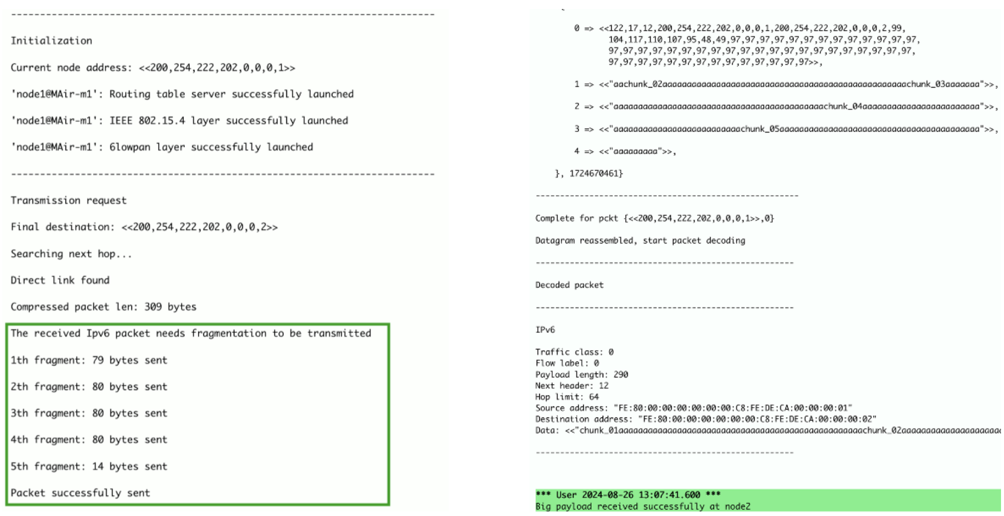


Figure 7.5: Big packet transmission

Observations As the simulation in Figs 7.5 shows, at the sender, the packet is too large to be transmitted as a single unit, so it is fragmented into five smaller parts before being sent. On the receiver side, each fragment received is stored. The first message confirms that all fragments of the packet with tag 0 from Node 1 have been successfully received. The packet is then reassembled and decoded, with the final output confirming that the decoded packet matches the original transmitted data. This confirms the expected behavior.

7.3.2.4 Meshed transmission

Description Transmission of an IPv6 packet that needs meshing. Node 1 is the sender, Node 3 the middle node, it forwards the packet and Node 2 the receiver.

Expected results Routing should be done correctly so that the packet reaches its destination, at the receiver it should successfully be decoded

Simulation output

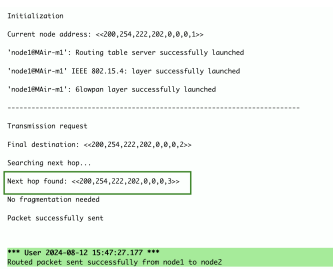


Figure 7.6: Meshed transmission Sender

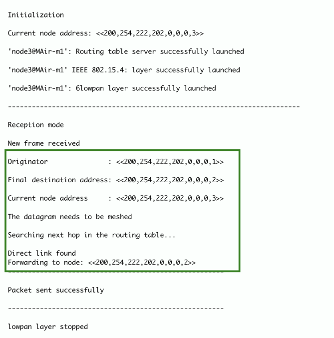


Figure 7.7: Meshed transmission Forwarder

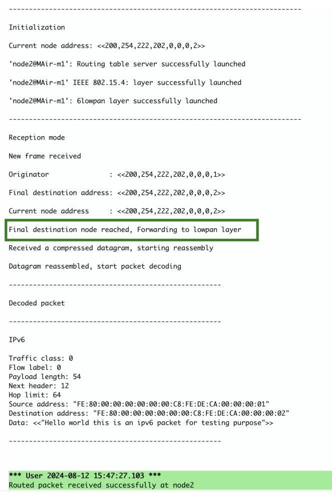


Figure 7.8: Meshed transmission Receiver

Observations As the simulation outputs shows, Node 1 first searches the next hop toward Node 2 in its routing table. The table is configured so that packets from Node 1 must pass through Node 3 to reach Node 2. A direct link is found between Node 1 and Node 3, so the packet is forwarded to Node 3. Upon receiving the packet, Node 3 detects that it is not the intended recipient and looks up the next hop in its routing table, finding a direct link to Node 2. The packet is then forwarded to Node 2. Once Node 2 receives the packet, it identifies itself as the intended recipient, reassembles, and decodes the packet. The decoded result matches the original transmission. This confirms the expected behavior.

7.3.2.5 Timeout

Description In this scenario, Node 1 sends only part of the data, causing a reassembly timeout at the receiver. Node 1 is the sender and Node 2 the receiver.

Expected results The sender should correctly transmit the first fragment, containing the data `hello`. the receiver, expecting `Hello world`, should trigger a reassembly timeout when `world` isn't received, discard the entry and returns a `reassembly_timeout` error.

Simulation output

```
Initialization

Current node address: <<200,254,222,202,0,0,0,1>>

'node1@MAir-m1': Routing table server successfully launched

'node1@MAir-m1' IEEE 802.15.4: layer successfully launched

'node1@MAir-m1': 6lowpan layer successfully launched

------------------------------------------------------------------------

Packet sent successfully


*** User 2024-08-12 15:47:50.062 ***
Incomplete payload sent from node1 to node2 to trigger a timeout
```

Figure 7.9: Timeout scenario Sender

```
DatagramMap after update:

{<<200,254,222,202,0,0,0,1>>,25} -> {datagram, 25, 12, 6,
    #{

        0 => <<"Hello ">>,

    }, 1723470469}


Incomplete first datagram, waiting for other fragments

Reassembly timeout for entry {<<200,254,222,202,0,0,0,1>>,25}

Entry deleted

*** User 2024-08-12 15:47:59.965 ***
Timeout occurred
```

Figure 7.10: Timeout scenario Receiver

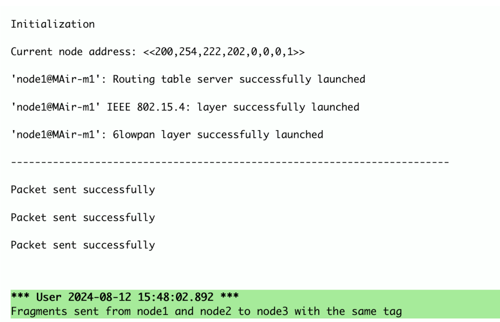
Observations The simulation output shows that the receiver successfully stored the first fragment with `hello`. However, since the second fragment isn't received, a timeout occurs, leading to the deletion of the entry, indicated by the `Entry deleted` message. This confirms the expected behavior.

7.3.2.6 Duplicate packet transmission

Description Transmission of duplicate fragment. Node 1 acts as the sender and Node 2 the receiver.

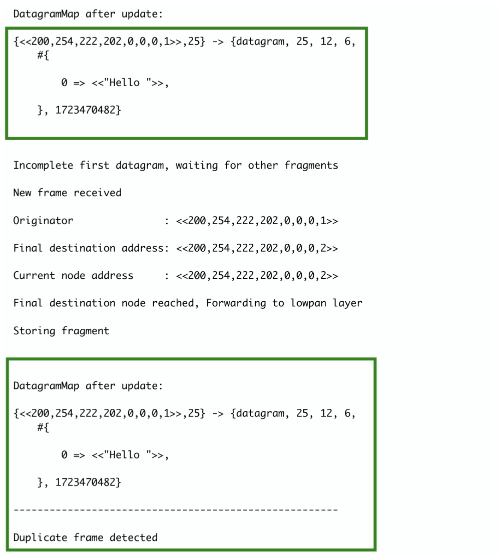
Expected results Node 1 should correctly send twice a fragment containing the hello payload. When the duplicated fragment is received at node 2, the latter should detect that it is a duplicate, and only keep the first fragment.

Simulation output



```
Initialization
Current node address: <<200,254,222,202,0,0,0,1>>
'node1@MAir-m1': Routing table server successfully launched
'node1@MAir-m1' IEEE 802.15.4: layer successfully launched
'node1@MAir-m1': 6lowpan layer successfully launched
------------------------------------------------------------------------
Packet sent successfully
Packet sent successfully
Packet sent successfully

*** User 2024-08-12 15:48:02.892 ***
Fragments sent from node1 and node2 to node3 with the same tag
```

Figure 7.11: Duplicate transmission Sender



```
DatagramMap after update:
{<<200,254,222,202,0,0,0,1>>,25} -> {datagram, 25, 12, 6,
    #{
        0 => <<"Hello ">>,
    }, 1723470482}

Incomplete first datagram, waiting for other fragments
New frame received
Originator              : <<200,254,222,202,0,0,0,1>>
Final destination address: <<200,254,222,202,0,0,0,2>>
Current node address     : <<200,254,222,202,0,0,0,2>>
Final destination node reached, Forwarding to lowpan layer
Storing fragment

DatagramMap after update:
{<<200,254,222,202,0,0,0,1>>,25} -> {datagram, 25, 12, 6,
    #{
        0 => <<"Hello ">>,
    }, 1723470482}
----------------------------------------------------
Duplicate frame detected
```

Figure 7.12: Duplicate transmission Receiver

Observations The output shows that 3 packets are sent on the sender's side, corresponding to the fragment containing the messages `hello, hello, world`. On the receiver side, after receiving the first fragment, it is stored and the node waits to receive the second. When the second fragment is received, a duplicate frame is detected and ignored.

7.4 Hardware tests

In this section, we'll take a look at hardware testing, including the setup used for the tests, the general workflow, and the actual tests performed on GRiSP 2 boards. The aim of these tests was firstly to validate the 6LoWPAN packet format using Wireshark software. Secondly, to ensure that the GRiSP2 boards communicate correctly with each other, in different scenarios and environments.

During the period of this thesis, there were only 5 prototype boards of the Pmod UWB sensors manufactured by Stritzinger Gmbh, and for the majority of my work, I only had access to 3 Pmod UWBs.

7.4.1 Robot application

To conduct the tests, the Erlang application behavior `robot` was developed to enable communication between GRiSP boards using the `lowpan api` module and Pmod UWB sensors. The `robot` application can be independently started, stopped, and configured, ensuring flexible communication within the OTP system. [42] The full code of the robot application is available in the appendix.

7.4.2 Testing environment setup

This sub-section presents the parameters to be defined in order to perform the tests properly.

7.4.2.1 Sniffer configuration

The tables below shows the parameters used to configure the sniffer.

Channel	PRF	Preamble length	Data rate	Preamble code
5	16MHz	1024	6.8 Mbps	4

PAC size	Standard frame delimiter	PHR	CRC filter
8	Standard	Standard	Off

Table 7.3: Sniffer configuration settings

7.4.2.2 Wireshark configuration

Packet analysis was made possible using version 4.2.1 of Wireshark. Note that when analyzing packets in Wireshark, the 6LoWPAN section may not appear

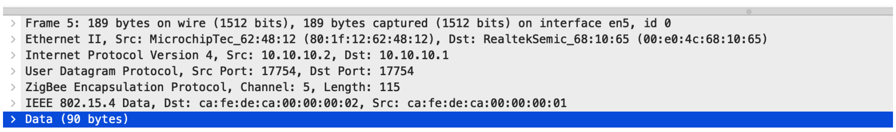


Figure 7.13: Wireshark setup 1

The 6LoWPAN protocol should to be enable for this section to appear. To do so, we need to go to the **Analyze** tab, then **Enabled protocols** and type 6LoWPAN in the search bar, then enable it.

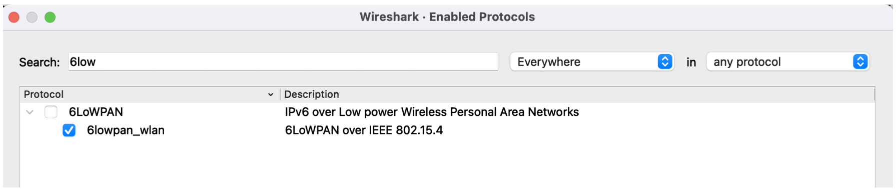


Figure 7.14: 6LoWPAN wirehsark activation

After these steps, the 6LoWPAN section should then appear, as shown in the next figure.

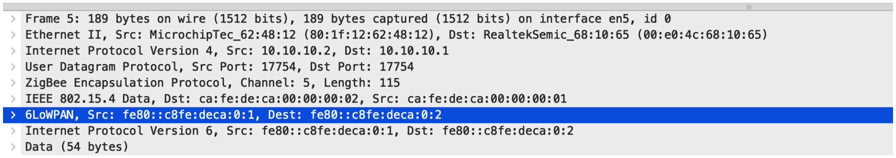


Figure 7.15: Wireshark setup 2

7.4.2.3 Workflow

The hardware test workflow was set up as illustrated in Figure 7.16. The GRiSP 2 board is connected to a PC via a serial connection, the UWB sniffer via an Ethernet connection and Wireshark runs on the laptop. When the Pmod UWB sensor on the GRiSP 2 board sends data over wireless communication, the sniffer captures these transmissions. The captured data is then transmitted to the laptop, where Wireshark displays the information in a user-friendly format, allowing for easy analysis of the communication.

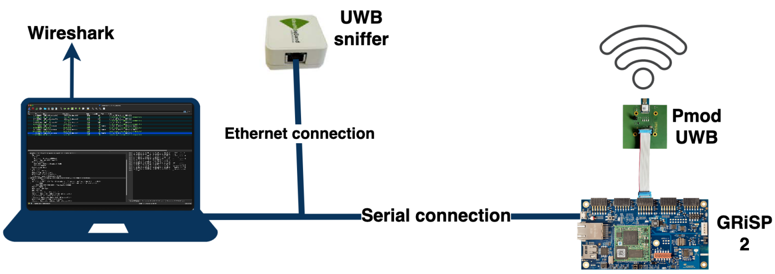


Figure 7.16: Hardware test workflow

7.4.3 Packet format validation

This section examines the Wireshark decoding results for various 6LoWPAN datagrams, covering all possible encapsulations. The primary goals were to confirm that Wireshark correctly identified the packets as 6LoWPAN and to verify accurate decoding of header fields and payloads. For each case, I compared the transmitted packet fields with those decoded in Wireshark to ensure a perfect match. A generic packet, with values listed in Table 7.1 was used for most transmissions, with adjustments made for specific test cases.

7.4.3.1 Uncompressed IPv6 datagram

Encapsulation type Uncompressed IPv6 datagram

Wireshark results

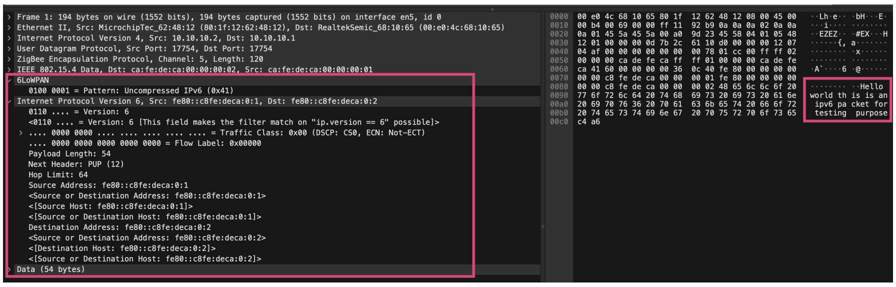


Figure 7.17: Uncompressed IPv6 datagram encapsulation, Wireshark pcap

Observations As Figure 7.17 shows, the packet is correctly identified as a 6LoWPAN packet, with the header pattern `0100 0001` confirming it as an uncompressed IPv6 packet. The inferred IPv6 values match the test packet, and the payload is accurately decoded.

7.4.3.2 Compressed IPv6 datagram

Encapsulation type Compressed IPv6 datagram

Wireshark results

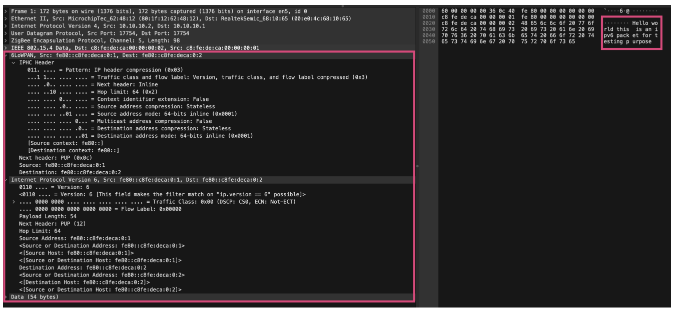


Figure 7.18: Compressed IPv6 datagram encapsulation, Wireshark pcap

Observations As shown in Figure 7.17, the IPHC Header confirms that Wireshark correctly identified the packet as compressed. The IPv6 section values match those defined for the test packet. The `Next header` value is carried in-line, which is expected since no compression scheme was defined for it (value 12). The 64-bit source and destination MAC addresses are correctly in-line, as no context was specified. The `M` (multicast) field is set to false, indicating a non-multicast destination address, consistent with the use of link-local addresses.

7.4.3.3 Meshed IPv6 datagram

Encapsulation type Compressed IPv6 datagram that requires mesh addressing

Wireshark results

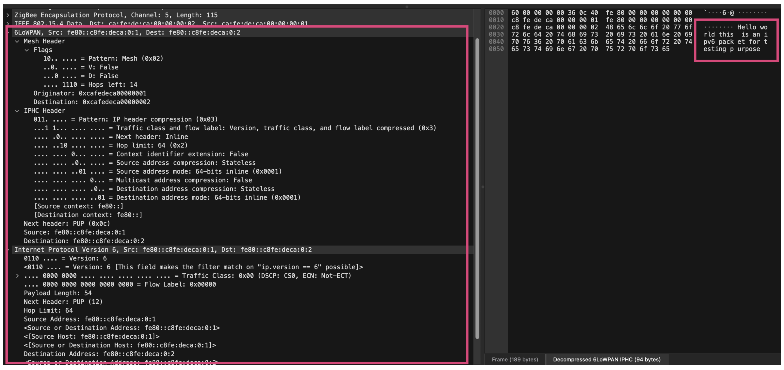


Figure 7.19: Meshed IPv6 datagram encapsulation, Wireshark pcap

Observations The Wireshark capture shows that the packet was correctly recognized as a compressed one requiring mesh networking, as seen under the 6LoWPAN section. The inferred values match the defined parameters. The V and F flags are set to false, indicating 64-bit MAC addresses in the Originator and Destination fields. The difference between these addresses and the last 64 bits of the IPv6 addresses is expected, as the IPv6 addresses were generated statelessly from the MAC addresses, with the `U/L` bit shifted.

7.4.3.4 Fragmented IPv6 datagram

Encapsulation type Compressed IPv6 datagram that requires fragmentation

Wireshark results

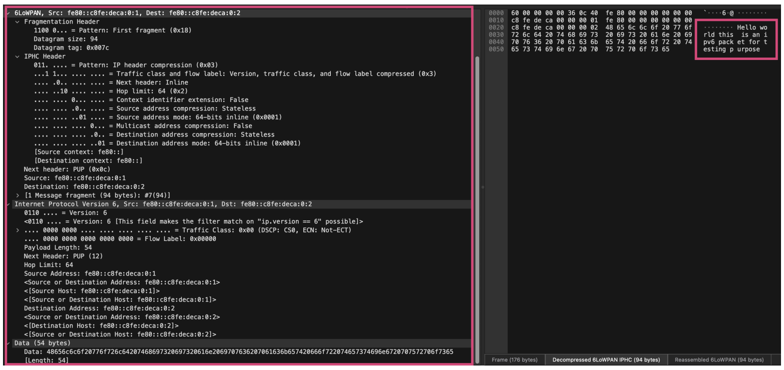


Figure 7.20: Fragmented IPv6 datagram encapsulation, Wireshark pcap

Observations As shown Figure 7.20, Wireshark correctly detected that this was a compressed and fragmented packet. Once again, the inferred values match those previously defined. We observe that the bit sequence of the fragment header begins with `1100 0...`, which corresponds to the dispatch value of first fragment header, as expected.

7.4.3.5 Meshed, fragmented and compressed IPv6 datagram

Encapsulation type Compressed IPv6 datagram that requires both mesh addressing and fragmentation

Wireshark results

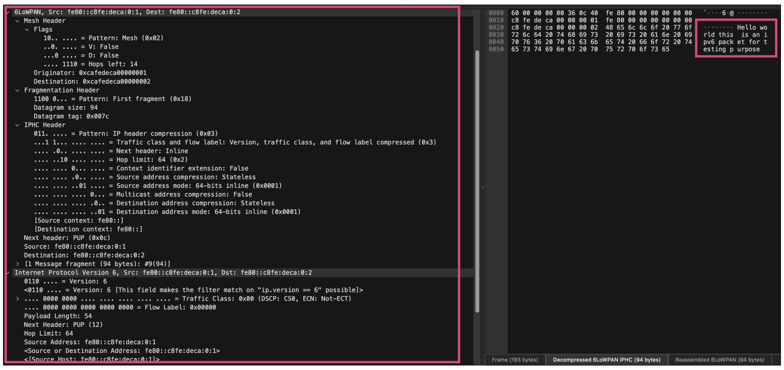


Figure 7.21: Compressed, meshed and fragmented encapsulation, Wireshark pcap

Observations The packet decoded in Wireshark shown in Figure 7.21, accurately corresponds to one that has been compressed, fragmented, and meshed. The order in which the different headers appear is also correctly maintained. The inferred fields values match the expected ones.

7.4.3.6 Broadcasted IPv6 datagram

Encapsulation type Compressed IPv6 datagram that requires both mesh addressing and a broadcast header to support mesh broadcast/multicast

Wireshark results

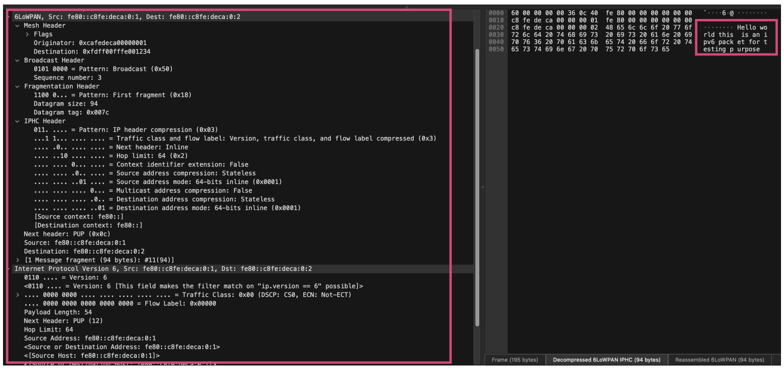


Figure 7.22: Broadcasted IPv6 datagram encapsulation, Wireshark pcap

Observations As expected, Wireshark shows that this packet required both meshing and broadcasting. Note the bit sequence `01010000` that corresponds to the dispatch value of the broadcast header. The inferred values are indeed the expected ones.

7.4.3.7 UDP next header

Encapsulation type Compressed IPv6 datagram with UDP as next header

Wireshark results

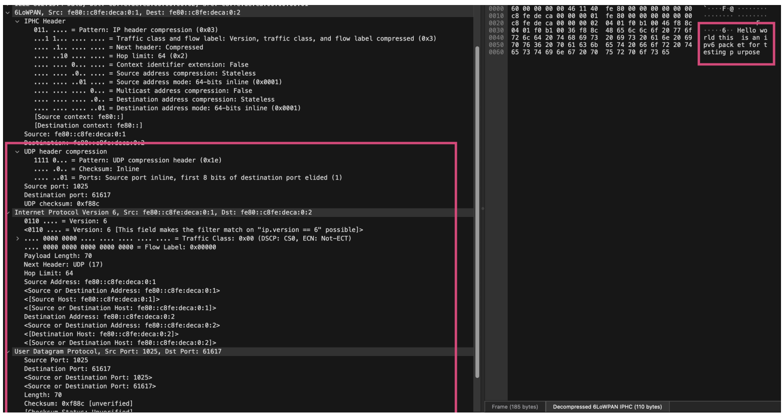


Figure 7.23: UDP encapsulation, Wireshark pcap

Observations The following figure shows the result of the compression when the UDP header follows the IPv6 header. The values of the inferred fields correspond to what was expected. Notice that this time the value of the Next Header field is correctly set to 17.

7.4.4 Two GRiSPs communication

This section presents the communication tests carried out between two GRiSP 2 boards.

7.4.4.1 Simple exchange

The first test was conducted to verify communication between the GRiSP 2 boards. A simple, non-fragmented packet was sent from node 1 (sender) to node 2 (receiver) over a direct link defined in the routing table. The communication results, captured from the serial terminals of the connected laptops, are shown in Figs 7.24 and 7.25 below.

Test condition The boards were placed 5 meters apart with no obstacles in between. The first node was positioned in a room with a WiFi router, while the second was in another room of the house.

Exchange output

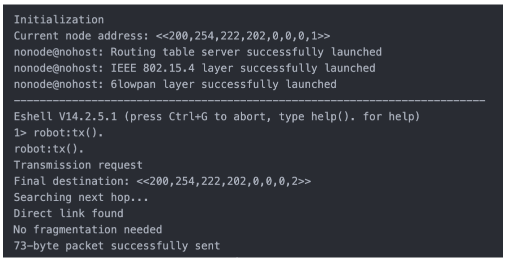


Figure 7.24: Simple communication GRiSP Sender

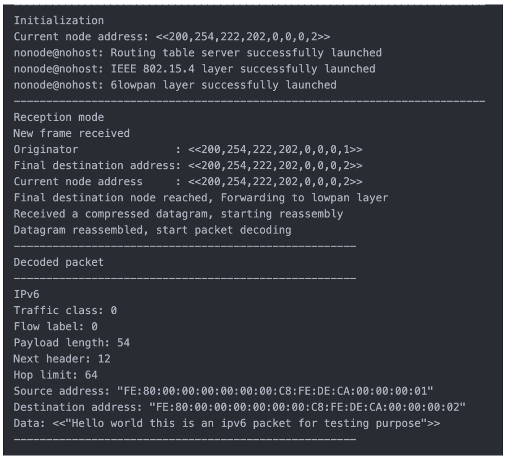


Figure 7.25: Simple communication GRiSP Receiver

Observations As shown in the captures, the boards behaved similarly to the simulation in Figs 7.3 and 7.4. After initialization, the `tx` function is called to send an IPv6 packet that doesn't require fragmentation. The board searches its routing table, finds a direct link to the destination, and sends the packet. On the receiver side, after initialization, it enters reception mode, waits for the frame, recognizes it's the intended recipient, and reassembles the packet.

7.4.4.2 Big packet exchange

The second test was about the transmission of a large 330-byte IPv6 packet.

Test condition The boards were placed 5 meters apart with no obstacles in between. The first node was positioned in a room with a WiFi router, while the second was in another room of the house.

Exchange output

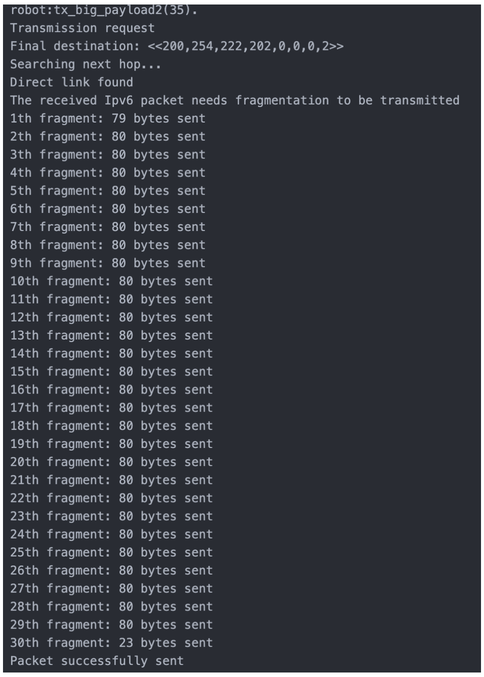


Figure 7.26: Big payload transmission GRiSP Sender

Figure 7.27: Big payload transmission GRiSP Receiver

Observations As shown in Figs 7.26 and 7.27, The `tx_big_payload` function is used to send a 290-byte compressed packet from the sender. The sender checks its routing table, finds a direct link to node 2, and fragments the packet into 21 parts for transmission. On the receiver side, each fragment is stored until all are received.

Once complete, the fragments are reassembled and decoded. This behavior was expected from the simulation described in sub-section 7.3.2.3.

7.4.4.3 Special case: Namur station

The final test between two boards took place at Namur station to assess performance in an interference-prone environment. The test involved sending multiple packet fragments while varying the distance between the nodes to observe if the exchange was successful.

Test condition At first, for short distances, the nodes were in the same glassed-in waiting room, then from 5m, they were separated by the room glass panes. The results of this exchange are shown in the table below.

Distance	Number of fragments	Transmission result
1m	1 to 25	Ok
2m	1 to 25	Ok
5m	1 to 25	Ok
8m	1 to 25	Ok
11m	1 to 25	Ok
16m	1 to 5	Partly lost
	> 5	lost

Table 7.4: Station communication tests

Observations The exchange went surprisingly well, it was only from a distance of 16 meters that I began to observe losses when more than five fragments were sent. These losses are due to the physical limitations of the Pmod UWB module. Notably, data losses and transmission errors occurred even at shorter distances when the receiver was positioned behind a wall. Tests conducted at 3 and 5 meters with the receiver behind a wall consistently resulted in transmission errors for the sender.

7.4.5 Three GRiSPs communication

The goal of the 3 board exchange was to verify correct routing. While successful board to board communication is important, the key is establishing a network where packets can route from an initiator to a destination node. Thus, routing tests are crucial. Two scenarios were defined, transmitting small and large packets to ensure both correctness and low transmission delay.

1st scenario In this scenario, node 1 was the initiator of the exchange, node 2 the forwarder and node 3 the receiver.

Test condition As shown in Figure 7.28, the 3 boards were positioned at the same level, with 4.80 m between node 2 and node 3, and 8.50 m between node 1 and node 3. Nodes 1 and 3 were out of each other's range and couldn't receive their respective packets. There was also a WiFi box in the room where the Node 2 was located.

2nd scenario In this scenario, node 1 was the initiator of the exchange, node 2 the forwarder and node 3 the receiver.

Test condition In this scenario, node 2 and node 3 were at the ground floor and node 1 at the first floor. A distance of 5.40m separated node 1 and node 2 and a distance of 4.80 m between node 2 and node 3. Nodes 1 and 3 were out of each other's range and couldn't receive their respective packets.

Exchange output

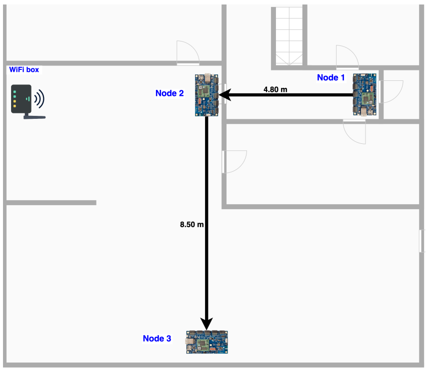


Figure 7.28: Three nodes routing test Sender

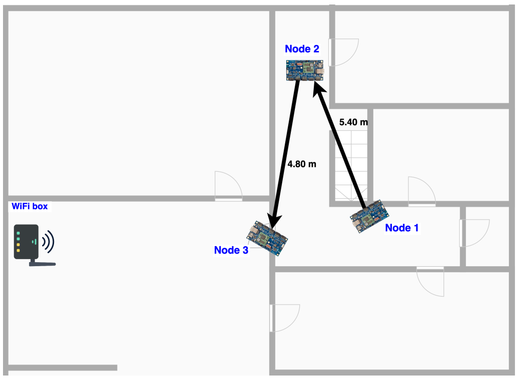


Figure 7.29: Three nodes routing test Receiver

Results

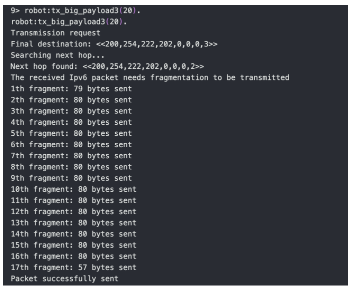


Figure 7.30: Routing test Sender

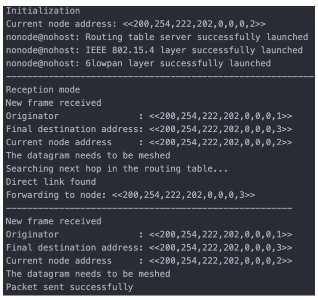


Figure 7.31: Routing test Forwarder

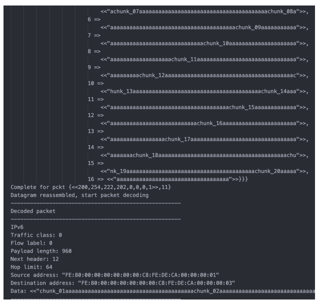


Figure 7.32: Routing test Receiver

Observations In this test, simple packet transmissions that did not require fragmentation were initially carried out to ensure that the boards communicated correctly, despite the presence of the Wi-Fi router, walls, and the difference in floors between the nodes (7.29). Next, exchanges of large packets requiring fragmentation and meshing were carried out. Figures 7.30, 7.31, and 7.32 show the results of serial terminal of each node.

As seen in Figure 7.30, Node 1 initiates the sending of a large packet to Node 3 by checking its routing table, finding Node 2 as the next hop, fragmenting the packet, and sending each fragment to Node 2. Figure 7.31 shows Node 2 forwarding the received fragments to Node 3 after identifying a direct link. Finally, Figure 7.32 illustrates Node 3 storing, reassembling, and decoding the fragments, confirming a successful transmission.

7.4.6 Five GRiSPs routing

In this scenario, node 1 was the initiator of the exchange, node 2 the forwarder and node 3 the receiver.

Test condition In the final stages of this thesis, additional tests were conducted using 5 GRiSP boards, which is the maximum number of nodes that could be equipped with Pmod sensors from Peer Stritzinger GmbH at the time of this thesis work, as the project is still in the prototype stage. The goal of these tests was to further confirm the effectiveness of the routing. The boards setup is given in Figure 7.33.

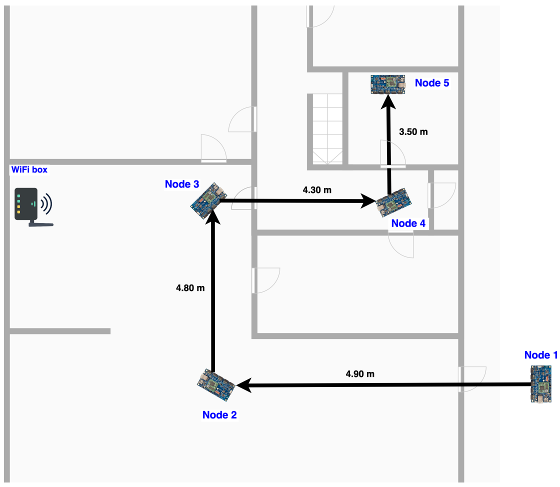


Figure 7.33: Five nodes routing test

Terminal outputs

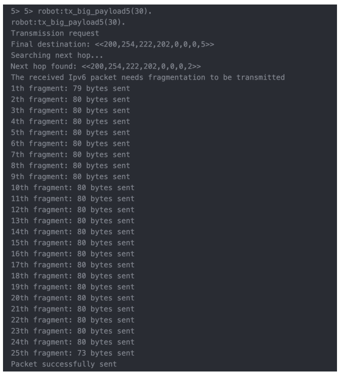


Figure 7.34: Five nodes routing Node 1

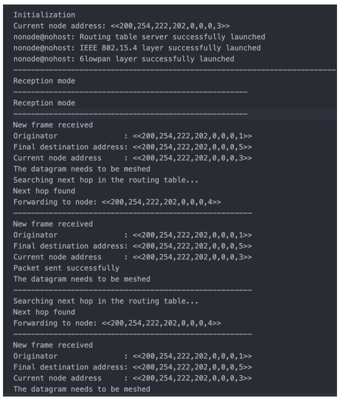


Figure 7.35: Five nodes routing Node 3

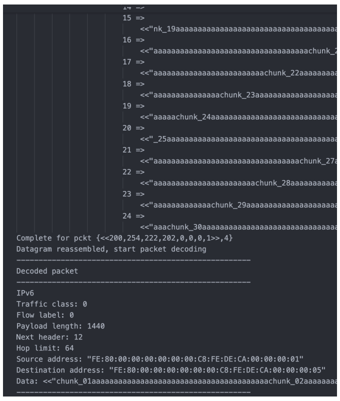


Figure 7.36: Five nodes routing Node 5

Observations The screenshots in Figures 7.34, 7.35, 7.36, show the terminals of nodes 1, 3, and 5. The behavior is identical to the scenario with 3 boards, with the difference being that the terminal node is now Node 5, and Node 3 is one of the forwarders. Notably, in Figure 7.35, we can see that when Node 3 receives a fragment from node 2, it checks its routing table for the next hop and finds node 4. At node 5, we see that not only are all the fragments received reassembled and decoded, but also that they come from node 1. This validates the test and confirms the expected behavior.

7.5 Contiki-ng validation tests

Additional tests were performed using the Cooja simulation software from Contiki-ng because it integrates 6LoWPAN. Cooja is a network simulator used for testing and developing IoT applications. The purpose of these tests was to compare the compression results of the implementation from Contiki-ng with the current implementation for a specific IPv6 packet.

7.5.1 Setup

To achieve this, a simulation of the UDP border router was conducted in Cooja 7.37 and while the simulation was running, the packet capture tool in Cooja was used to capture the exchanged packets. These packets were then analyzed in Wireshark. The IPv6 values inferred by Wireshark were subsequently used to create IPv6 packets, which were then transmitted to the `lowpan api` to perform the compression operation.

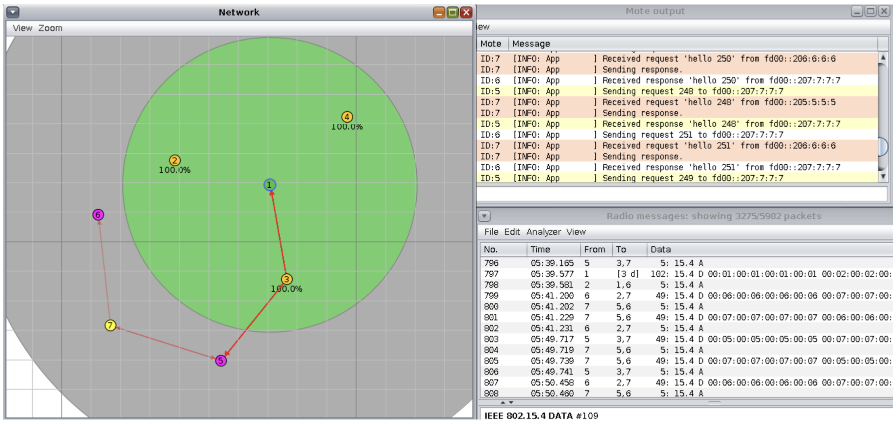


Figure 7.37: UDP border router cooja simulation

7.5.2 First example

The packet field values transmitted in this first test are shown in the table

Field	**value**
Traffic class	0
Flow label	0
Next header	58
Hop limit	64
Source address	`FE:80::207:7:7:7`
Destination address	`FF:02::1a`

Table 7.5: 1st Contiki example IPv6 field values

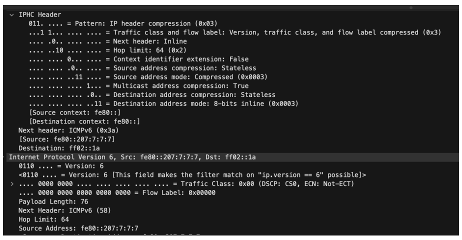


Figure 7.38: Contiki 1st example Wireshark capture

```
Expected carried values: #{"DAM" => <<26>>,"NextHeader" => 58}

Actual carried values: #{"DAM" => <<26>>,
                         "NextHeader" => 58,
                         "SAM" => <<0,207,0,7,0,7,0,7>>}
```

Figure 7.39: 1st example compression output

Observations I expected the IP packet's Next Header (58, ICMPv6) and the last 8 bits of the destination address (formatted as `ff02::00XX`) to be carried in-line, along with the last 64 bits of the source address. Figure 7.39 shows that Contiki meets these expectations but unexpectedly elides the source address.

7.5.3 Second example

The packet field values transmitted in this second test are shown in the table

Field	value
Traffic class	0
Flow label	0
Next header	17
Hop limit	64
Source address	FE:80::202:2:2:2
Destination address	FE:80::212:7402:2:2
UDP source port	5683
UDP destination port	5683
UDP checksum	8441

Table 7.6: 2nd Contiki example IPv6 field values

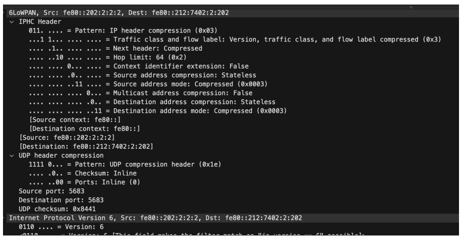


Figure 7.40: Contiki 2nd example Wireshark capture

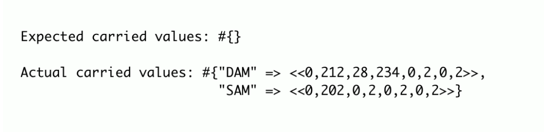


Figure 7.41: 2nd example compression output

Observations The Figure 7.41 shows the IPv6 compression result from both Contiki and current implementation. I expected the last 64 bits of the source and destination addresses to be carried in-line, but Figure 7.41 shows that Contiki compressed all fields.

7.5.4 Third example

The packet field values transmitted in this third test are shown in the table

Field	value
Traffic class	0
Flow label	0
Next header	43
Hop limit	63
Source address	::207:7:7:7
Destination address	::202:2:2:2

Table 7.7: 3rd Contiki example IPv6 field values

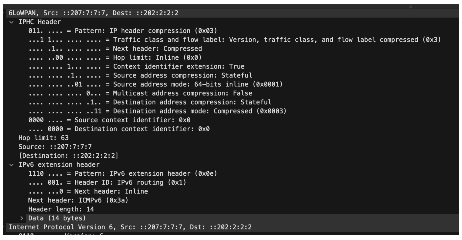


Figure 7.42: Contiki 3rd example Wireshark capture

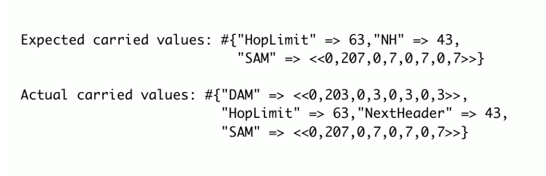


Figure 7.43: 3rd example compression output

Observations I expected the last 64 bits of the source and destination addresses, the Next Header (43, Routing header), and the Hop Limit to be carried in-line. Figure 7.43 shows that in Contiki, the Hop Limit, Next Header, and source address are carried in-line, but the destination address is elided.

7.5.5 Observations

The Contiki-ng tests showed only a partial match between the proposed implementation and Contiki's compression. Contiki often behaves as if a predefined context exists, leading to the omission of bits from source/destination addresses.

Although the tests were not entirely conclusive, the analysis led to an optimization: when a packet is meshed, the last 64 bits of the initiator's and recipient's addresses, found in the mesh header, can be elided since they can be restored at the destination even without context.

Chapter 8

Future work

Although the current implementation has shown satisfying results, several aspects can be improved to enhance the performance.

Firstly, the compression scheme used in the implementation could be further optimized. Studies such as [43] and [44] have demonstrated better performance compared to the IPHC compression method introduced in the RFC6282. Implementing these advanced compression schemes could result in more efficient use of the resources in LoWPAN networks.

Furthermore, the routing logic defined in the current implementation is quite basic. It was developed using a simple `gen_server` allowing manual addition, deletion and update of routes, without any automated routing algorithm. For a more robust and scalable solution, the integration of established routing protocols is necessary. Specifically, two approaches exist for 6LoWPAN routing: the mesh-under approach and the route-over approach, the latter being more commonly deployed. Given Peer Stritzinger GmbH's objective to integrate the Thread standard into GRiSP 2 boards, route-over appears to be the most interesting choice, as it operates on the IP layer, aligning with the next stage after 6LoWPAN implementation in the Thread stack.[11]. Furthermore, studies such as [45] and [46] have highlighted the benefits of RPL over mesh-under routing, particularly in terms of scalability, reliability, and reduced overhead in dense network environments.

Lastly, while the current implementation supports the compression of the UDP next header, adding compression for other next header cases such as TCP would be beneficial. Although the RFCs do not explicitly mandate compression for all next header types, extending the compression capabilities to include additional headers could improve performance, especially in diverse application scenarios.

Chapter 9

Conclusion

This thesis focused on the development and implementation of the 6LoWPAN protocol for GRiSP 2 embedded systems. The primary objective was to integrate essential 6LoWPAN mechanisms, including compression, fragmentation, and meshing, into the GRiSP environment. To accomplish this, a well-structured code architecture was designed in Erlang, enabling the GRiSP boards to efficiently handle IPv6 packets and ensuring seamless interaction between the 6LoWPAN layer and both the MAC and IP layers. The second phase involved rigorous testing to validate the correctness of these implementations, including functional tests, simulation scenarios, and real-world hardware evaluations. The results were positive, demonstrating the effectiveness of the implemented features and confirming that the GRiSP boards can now successfully exchange IPv6 packets while benefiting from 6LoWPAN functionalities.

Bibliography

[1] Stritzinger GmbH. *What is MTU ?* . august 2024. Accessed on 09-08-24, https://www.grisp.org/.

[2] Sewio. *UWB sniffer*. august 2024. Accessed on 09-08-24, https://www.sewio.net/wp-content/uploads/2018/07/UWBSniffer_datasheet_v0.2.pdf.

[3] Gauri Gupta Jitendra K. Pandey Surajit Mondal Oroos Arshi, Akanksha Rai. *IoT in energy: a comprehensive review of technologies, applications, and future directions*, june 2024. Accessed on 09-08-24, https://www.researchgate.net/publication/381163617_IoT_in_energy_a_comprehensive_review_of_technologies_applications_and_future_directions.

[4] statista. *Number of Internet of Things (IoT) connections worldwide from 2022 to 2023, with forecasts from 2024 to 2033*. *statista*, 2024. Accessed on 09-08-24, https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.

[5] IETF. *RFC 6568 - Design and Application Spaces for 6LoWPANs*, 2012. Accessed on 09-08-24, https://datatracker.ietf.org/doc/rfc6568/.

[6] IETF. *RFC 4919 - IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals*, 2007. Accessed on 09-08-24, https://datatracker.ietf.org/doc/rfc4919/.

[7] Md Turab Hossain. *A Review on IPv4 and IPv6: A comprehensive survey*. 2022. Accessed on 09-08-24, https://www.researchgate.net/publication/357584019_A_Review_on_IPv4_and_IPv6_A_comprehensive_survey.

[8] Wikipedia. *IPv6*. 2024. Accessed on 09-08-24, https://en.wikipedia.org/wiki/IPv6.

[9] Wikipedia. *Broadcast address*. July 2024. Accessed on 09-08-24, https://en.wikipedia.org/wiki/Maximum_transmission_unit.

[10] Wikipedia. *IEEE 802.15.4*. 2024. Accessed on 09-08-24, https://en.wikipedia.org/wiki/IEEE_802.15.4.

[11] Thread Group. *Thread Usage of 6LoWPAN*. 2015. Page 3, https://www.silabs.com/documents/public/white-papers/Thread-Usage-of-6LoWPAN.pdf.

[12] Thread Group. *Thread Usage of 6LoWPAN*. 2015. Page 6, https://www.silabs.com/documents/public/white-papers/Thread-Usage-of-6LoWPAN.pdf.

[13] Stritzinger GmbH. *An Erlang virtual machine on bare-metal board*. 2024. Accessed on 09-08-24, https://www.stritzinger.com/.

[14] Melanie Deputter. *UWB versus other tracking technologies in 2024*. 2024. Accessed on 09-08-24, https://www.pozyx.io/newsroom/uwb-versus-other-technologies.

[15] Wikipedia. *Ultra-wideband*. 2024. Accessed on 09-08-24, https://en.wikipedia.org/wiki/Ultra-wideband.

[16] Open Thead. *What is Thread?* 2024. Accessed on 09-08-24, https://openthread.io/guides/thread-primer.

[17] Jonathan Affrye. *6LoWPAN for UWB communication on the GRiSPs boards*. 2024. Accessed on 09-08-24, https://github.com/jonaffrye/6lowpan.

[18] David P. Williams and Amir Rasouli. *A Survey of Indoor Location Technologies, Techniques and Applications in Industry*. *Journal of Manufacturing Systems*, 63:317–332, 2022. Accessed on 09-08-24, https://www.sciencedirect.com/science/article/pii/S2542660522000907.

[19] Ziyu Zhang, Keyu Li, and Bo Wang. *Indoor Location Technology with High Accuracy Using Simple Visual Tags*. *ResearchGate*, 2023. Accessed on 09-08-24, https://www.researchgate.net/publication/368230876_Indoor_Location_Technology_with_High_Accuracy_Using_Simple_Visual_Tags.

[20] *Concurrency in Python: How to Overcome Challenges and Debugging Techniques*. May 26, 2023. Accessed on 09-08-24, https://decodepython.com/concurrency-in-python-how-to-overcome-challenges-and-debugging-techniques/.

[21] dynatrace. *The Impact of Garbage Collection on Application Performance*. 2024. Accessed on 09-08-24, https://www.dynatrace.com/resources/ebooks/javabook/impact-of-garbage-collection-on-performance/.

[22] Rust. *Understanding Ownership*. 2024. Accessed on 09-08-24, https://doc.rust-lang.org/book/ch04-00-understanding-ownership.html.

[23] Joe Armstrong. *Making reliable distributed systems in the presence of software errors*. 2003. Accessed on 09-08-24, https://erlang.org/download/armstrong_thesis_2003.pdf.

[24] Francesco Cesarini and Simon Thompson. Erlang Programming. 2009. Accessed on 09-08-24, https://www.academia.edu/35383939/Erlang_programming_main.

[25] Peer Stritzinger. *GRiSP 2*. 2019. Accessed on 09-08-24, https://www.indiegogo.com/projects/grisp-2#/.

[26] Digilent. *Pmods*. 2024. Accessed on 09-08-24, https://digilent.com/reference/pmod/start.

[27] Qorvo. *DW1000 IEEE802.15.4-2011 UWB Transceiver*. 2017. Accessed on 09-08-24, https://www.qorvo.com/products/d/da007946.

[28] Sewio Networks. *UWB Sniffer Installation Guide*, 2023. Accessed on 09-08-24, https://www.sewio.net/uwb-sniffer/uwb-sniffer-installation/.

[29] Laurent Gwendal. *Ultra-Wideband for internet of things*. 2023. Accessed on 09-08-24, https://webperso.info.ucl.ac.be/~pvr/Laurent_20931800_2023.pdf.

[30] IETF. *IPv6 over Low power WPAN (6lowpan)*. Accessed on 09-08-24, https://datatracker.ietf.org/wg/6lowpan/about/.

[31] IETF. *RFC 4944 - Transmission of IPv6 Packets over IEEE 802.15.4 Networks*, 2007. Accessed on 09-08-24, https://datatracker.ietf.org/doc/rfc4944/.

[32] IETF. *RFC 6282 - Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks*, 2011. Accessed on 09-08-24, https://datatracker.ietf.org/doc/rfc6282/.

[33] IETF. *RFC 6606 - Problem Statement and Requirements for a 6LoWPAN Router*, 2012. Accessed on 09-08-24, https://datatracker.ietf.org/doc/rfc6606/.

[34] IETF. *RFC 6775 - Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)*, 2012. Accessed on 09-08-24, https://datatracker.ietf.org/doc/rfc6775/.

[35] Transmission of IPv6 Packets over Ethernet Networks. *IEEE 802.15.4*. 2024. Page 3, https://datatracker.ietf.org/doc/html/rfc2464.

[36] Wikipedia. *MAC address*. July 2024. Accessed on 09-08-24, https://en.wikipedia.org/wiki/MAC_address.

[37] ErlangOTP. *Design principles*. August 2024. Accessed on 09-08-24, https://www.erlang.org/doc/system/design_principles.html.

[38] Erlang. *gen_statem Behavior*. august 2024. Accessed on 09-08-24, http://erlang.org/~raimo/gen_statem-enter-event/doc/design_principles/statem.html#Event%20Types.

[39] Erlang documentation. *gen_server Behaviour*. August 2024. Accessed on 09-08-24, https://www.erlang.org/doc/system/gen_server_concepts.html.

[40] Erlang documentation. *Common Test Basics*. August 2024. Accessed on 09-08-24, https://www.erlang.org/doc/apps/common_test/basics_chapter.html.

[41] 6LoWPAN Examples of the IPv6 Header Compression. *COMNET Protocols*. August 2024. Accessed on 09-08-24, https://www.youtube.com/watch?v=0JMV03HN0xo.

[42] Erlang documentation. *application behaviour*. August 2024. Accessed on 09-08-24, https://www.erlang.org/doc/apps/kernel/application.html.

[43] Asif Rahmani, Mohammad Farsi, Babak Javadi, and Jagannath Singh. *Modified and Improved IPv6 Header Compression (MIHC) Scheme for 6LoWPAN*. 2018. Accessed on 09-08-24, https://www.researchgate.net/publication/325659788_Modified_and_Improved_IPv6_Header_Compression_MIHC_Scheme_for_6LoWPAN.

[44] Geetha V.; Archana Bhat; S. Thanmayee. *New Bit Pattern Based IPv6 Address Compression Techniques for 6LoWPAN Header Compression*. 2015. Accessed on 09-08-24, https://ieeexplore.ieee.org/document/9837064.

[45] Hyon-Soo Cha Hassen Redwan S.M. Saif Shams Ki-Hyung Kim Seung-Wha Yoo Aminul Haque Chowdhury, Muhammad Ikram. *Route-over vs Mesh-under Routing in 6LoWPAN*, 2009. Accessed on 09-08-24, https://www.researchgate.net/publication/220762351_Route-over_vs_mesh-under_routing_in_6LoWPAN.

[46] Bernard Tourancheau Malisa Vucinic and Andrzej Duda. *Performance Comparison of the RPL and LOADng Routing Protocols in a Home Automation Scenario*. Accessed on 09-08-24, https://arxiv.org/pdf/1401.0997.

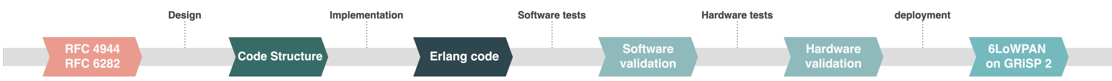
Appendix A

Appendix

A.1 General Note

Artificial intelligence tools DeepL and ChatGPT were used to improve the quality of both English and text.

A.2 Methodology roadmap



A.3 Lowpan header file code

```
% @doc 6LoWPAN header
-include("ieee802154.hrl").
-include("mac_frame.hrl").

%-----------------------------------------------------------------------------
% Useful records
%-----------------------------------------------------------------------------
-record(ipv6PckInfo,
        {version = 6,
         trafficClass,
         flowLabel,
         payloadLength,
         nextHeader,
         hopLimit,
         sourceAddress,
         destAddress,
         payload}).
-record(datagramInfo, {fragtype, datagramSize, datagramTag, datagramOffset, payload}).


```

```erlang
%--------------------------------------------------------------------------------
% Dispatch Type and Header
%--------------------------------------------------------------------------------

%@doc dispatch value bit pattern from rfc4944, DH stands for dispatch header

-define(NALP_DHTYPE, 2#00). % Not a LoWPAN frame, such packet shall be discarded
-define(IPV6_DHTYPE, 2#01000001). % Uncompressed IPv6 Addresses
-define(IPHC_DHTYPE, 2#011).      %  LOWPAN_IPHC compressed IPv6 (RFC6282)
-define(BC0_DHTYPE, 2#01010000). % LOWPAN_BC0 broadcast
-define(ESC_DHTYPE, 2#01111111). % Additional Dispatch byte follows
-define(MESH_DHTYPE, 2#10). % Mesh Header
-define(FRAG1_DHTYPE, 2#11000). % Frist fragmentation Header
-define(FRAGN_DHTYPE, 2#11100). % Subsequent fragmentation Header
-define(UDP_DHTYPE, 2#11110). % UDP header compression

-define(0xf0b, 2#111100001011).
-define(0xf0, 2#11110000).

-type dispatch_type() ::
    ?NALP_DHTYPE |
    ?IPV6_DHTYPE |
    ?IPHC_DHTYPE |
    ?BC0_DHTYPE  |
    ?ESC_DHTYPE  |
    ?MESH_DHTYPE |
    ?FRAG1_DHTYPE|
    ?FRAGN_DHTYPE.

%--------------------------------------------------------------------------------
% Fragmentation Type and Header
%--------------------------------------------------------------------------------

-type frag_type() :: ?FRAG1_DHTYPE | ?FRAGN_DHTYPE.

-record(frag_header,
        {frag_type = ?FRAG1_DHTYPE :: frag_type(),
         datagram_size, % 11 bits
         datagram_tag, % 16 bits
         datagram_offset}). % 8-bits

-record(frag_info,
        {datagram_size,
         datagram_tag,
         datagram_offset
        }).

-record(datagram,
        {tag,
         size,
         cmpt,
         timer,
         fragments
         }).

-define(MAX_FRAME_SIZE,80). % Because IEEE 802.15.4 leaves approximately 80-100
    bytes of payload
-define(MAX_FRAG_SIZE_NoMESH,75). % Because max frame size is 80 bytes, and lowpan
     header 30 bytes (5 bytes for fragHeader) 8 bytes are from IPHC which is
    included in payload for frag
-define(MAX_FRAG_SIZE_MESH,58). % Considering max frame size = 80 bytes, lowpan
    header = 30 bytes (17 bytes for meshHeader, 5 bytes for fragHeader, 8 bytes
```

```
    for IPHC)
-define(MAX_DTG_SIZE, 2047). % 11 bits datagram_size
-define(REASSEMBLY_TIMEOUT, 60000). % 60 sec
-define(FRAG_HEADER_SIZE,5). % 5 bytes including frag_type, datagram_size,
    datagram_tag, and datagram_offset
-define(DATAGRAMS_MAP,#{}). % map of received datagrams, the keys are the tag of
    datagrams
-define(MAX_TAG_VALUE, 65535).
-define(DEFAULT_TAG_VALUE, 2#0000000000000000).
-define(BC_SEQNUM, 2#00000000).

-record(additional_info,
        {datagram_size,
         datagram_tag,
         hops_left,
         timer
        }).
-define(INFO_ON, 1).
-define(INFO_OFF, 0).

%-------------------------------------------------------------------------------
% Header Compression
%-------------------------------------------------------------------------------
-record(ipv6_header,
        {version = 2#0110, % 4-bit
         traffic_class, % 8-bit
         flow_label, % 20-bit
         payload_length, % 16-bit
         next_header, % 8-bit
         hop_limit, % 8-bit
         source_address, % 128-bit
         destination_address}). % 128-bit
-record(udp_header,
        {source_port, % 16-bit
         destination_port,  % 16-bit
         length,  % 16-bit
         checksum}). % 16-bit
-record(iphc_header,
        {dispatch = ?IPHC_DHTYPE, % 3-bit dispatch value
         tf, % 2-bit field for Traffic Class and Flow Control compression options
         nh, % 1-bit field for Next Header encoding using NHC
         hlim, % 2-bit field for Hop Limit compression
         cid, % 1-bit field for Context Identifier Extension
         sac, % 1-bit field for Source Address Compression (stateless or stateful)
         sam, % 2-bit field for Source Address Mode
         m, % 1-bit field for Multicast Compression
         dac, % 1-bit field for Destination Address Compression (stateless or
             stateful)
         dam}). % 2-bit field for Destination Address Mode

-define(LINK_LOCAL_PREFIX, 16#FE80).
-define(MULTICAST_PREFIX, 16#FF02).
-define(GLOBAL_PREFIX_1, 16#2001).
-define(GLOBAL_PREFIX_3, 16#2003).
-define(MESH_LOCAL_PREFIX, 16#FD00).
-type prefix_type() :: ?LINK_LOCAL_PREFIX | ?GLOBAL_PREFIX_1 | ?MULTICAST_PREFIX.


-define(UDP_PN, 17). % PN stands for Protocol Number
-define(TCP_PN, 6).
-define(ICMP_PN, 58).

```

```
% inspired from Thread Usage of 6LoWPAN
-define(Context_id_table,
        #{1 => <<?MESH_LOCAL_PREFIX:16,16#0DB8:16, 0:32>>, % mesh local prefix
          2 => <<0:64>>, % cooja mesh local prefix
          %2 => <<?GLOBAL_PREFIX_1:16, 0:48>>, % global prefix 1
          3 => <<?GLOBAL_PREFIX_3:16, 0:48>>}). % global prefix 3

-define(Prefixt_id_table,
        #{<<?MESH_LOCAL_PREFIX:16, 0:48>> => 1 , % mesh local prefix
        <<0:64>> => 2, % cooja mesh local prefix
       % <<?GLOBAL_PREFIX_1:16, 0:48>> => 2, % global prefix 1
        <<?GLOBAL_PREFIX_3:16, 0:48>> => 3}). % global prefix 3

-define(SHORT_ADDR_LEN, 2).
-define(EXTENDED_ADDR_LEN, 8).

%--------------------------------------------------------------------------------
% Routing
%--------------------------------------------------------------------------------

-define(BroadcastAdd, <<"    ">>).
-define(ACK_FRAME, <<>>).

-record(mesh_header,
        {mesh_type = ?MESH_DHTYPE,
         v_bit, % 1-bit
         f_bit, % 1-bit
         hops_left, % 4-bit
         originator_address, % link-layer address of the Originator
         final_destination_address  % link-layer address of the Final Destination
         %deep_hops_left = undefined
        }).

-record(meshInfo,
        {version = 6,
         v_bit,
         f_bit,
         hops_left,
         originator_address,
         final_destination_address,
         deep_hops_left,
         payload}).


-define(Max_Hops, 2#1110).
-define(DeepHopsLeft, 16#F). % 0xF
-define(Max_DeepHopsLeft, 2#11111111). % 8-bit Deep Hops Left

-define(Node1MacAddress, <<16#CAFEDECA00000001:64>>).
-define(Node2MacAddress, <<16#CAFEDECA00000002:64>>).
-define(Node3MacAddress, <<16#CAFEDECA00000003:64>>).
-define(Node4MacAddress, <<16#CAFEDECA00000004:64>>).
-define(Node5MacAddress, <<16#CAFEDECA00000005:64>>).
```

```
% Used to test 16-bit node addresses
% -define(Node1MacAddress, <<16#0001:16>>).
% -define(Node2MacAddress, <<16#0002:16>>).
% -define(Node3MacAddress, <<16#0003:16>>).


-define(node1_addr,
        lowpan_core:generateEUI64MacAddr(?Node1MacAddress)).
-define(node2_addr,
        lowpan_core:generateEUI64MacAddr(?Node2MacAddress)).
-define(node3_addr,
        lowpan_core:generateEUI64MacAddr(?Node3MacAddress)).
-define(node4_addr,
        lowpan_core:generateEUI64MacAddr(?Node4MacAddress)).
-define(node5_addr,
        lowpan_core:generateEUI64MacAddr(?Node5MacAddress)).

-define(Default_routing_table,
        #{?node1_addr => ?node1_addr,
          ?node2_addr => ?node2_addr,
          ?node3_addr => ?node3_addr,
          ?node4_addr => ?node4_addr,
          ?node5_addr => ?node5_addr}).

% -define(Node1_routing_table,
%          #{?node1_addr => ?node1_addr,
%            ?node2_addr => ?node3_addr,
%            ?node3_addr => ?node2_addr}).

-define(Node1_routing_table, % 5 node routing test: 1 -> 2 -> 5
        #{?node5_addr => ?node2_addr,
          ?node2_addr => ?node2_addr,
          ?node3_addr => ?node3_addr,
          ?node4_addr => ?node4_addr}).

-define(Node2_routing_table, % 5 node routing test: 2 -> 3 -> 5
        #{?node5_addr => ?node3_addr}).

-define(Node3_routing_table, % 5 node routing test: 3 -> 4 -> 5
        #{?node5_addr => ?node4_addr}).

-define(Node4_routing_table, % 5 node routing test: 4 -> 5
        #{?node5_addr => ?node5_addr}).

-define(Node5_routing_table, % 5 node routing test
        #{?node5_addr => ?node5_addr}).





%---------------------------------------------------------------------------------
% Metrics
%---------------------------------------------------------------------------------
-record(metrics,
        {ack_counter = 0,
        fragments_nbr = 1,
        start_time = 0,
        end_time = 0,
        pckt_len = 0,
        compressed_pckt_len = 0}).
```

A.4 Lowpan core code

```
-module(lowpan_core).

-include("lowpan.hrl").

-export([
    pktEncapsulation/2, fragmentIpv6Packet/3,
    reassemble/1, storeFragment/8, createIphcPckt/2, getIpv6Pkt/2, datagramInfo/1,
    compressIpv6Header/2, buildDatagramPckt/2, buildFirstFragPckt/5,
    getPcktInfo/1, getIpv6Payload/1, triggerFragmentation/3,
    decodeIpv6Pckt/4, encodeInteger/1,
    tupleToBin/1, buildFragHeader/1, getNextHop/6,
    generateChunks/0, generateChunks/1,
    buildMeshHeader/1, getMeshInfo/1, containsMeshHeader/1,
    buildFirstFragHeader/1, getUncIpv6/1, getEUI64From48bitMac/1, generateLLAddr/1,
    getEUI64MacAddr/1, createNewMeshHeader/3, createNewMeshDatagram/3, removeMeshHeader/2,
    convertAddrToBin/1,checkTagUnicity/2, get16bitMacAddr/1, generateMulticastAddr/1,
    getDecodeIpv6PcktInfo/1, getNextHop/2, generateEUI64MacAddr/1
]).
-export([getEUI64FromShortMac/1]).
-export([getEUI64FromExtendedMac/1]).

%-------------------------------------------------------------------------------
%% @doc Returns an Ipv6 packet
%% @spec getIpv6Pkt(Header, Payload) -> binary().
%-------------------------------------------------------------------------------
-spec getIpv6Pkt(Header, Payload) -> binary() when
      Header :: binary(),
      Payload :: binary().
getIpv6Pkt(Header, Payload) ->
    ipv6:buildIpv6Packet(Header, Payload).

%-------------------------------------------------------------------------------
%% @doc create an uncompressed IPv6 packet
%% @spec pktEncapsulation(Header, Payload) -> binary().
%-------------------------------------------------------------------------------
-spec pktEncapsulation(Header, Payload) -> binary() when
      Header :: binary(),
      Payload :: binary().
pktEncapsulation(Header, Payload) ->
    Ipv6Pckt = getIpv6Pkt(Header, Payload),
    DhTypebinary = <<?IPV6_DHTYPE:8, 0:16>>,
    <<DhTypebinary/binary, Ipv6Pckt/binary>>.

%-------------------------------------------------------------------------------
%% @doc Encapsulates an Uncompressed IPv6 packet
%% @spec getUncIpv6(Ipv6Pckt) -> binary().
%-------------------------------------------------------------------------------
-spec getUncIpv6(Ipv6Pckt) -> binary() when
      Ipv6Pckt :: binary().
getUncIpv6(Ipv6Pckt) ->
    <<?IPV6_DHTYPE:8, Ipv6Pckt/bitstring>>.
```

```

%-------------------------------------------------------------------------------
%
%                               Header compression
%
%-------------------------------------------------------------------------------

%-------------------------------------------------------------------------------
%% @doc Compresses an Ipv6 packet header according to the IPHC compression scheme
%% @spec compressIpv6Header(Ipv6Pckt, RouteExist) -> {binary(), map()}.
%% @returns a tuple containing the compressed header in binary form
%% and the values that should be carried inline
%-------------------------------------------------------------------------------
-spec compressIpv6Header(Ipv6Pckt, RouteExist) -> {binary(), map()} when
      Ipv6Pckt :: binary(),
      RouteExist :: boolean().
compressIpv6Header(Ipv6Pckt, RouteExist) ->
    PcktInfo = getPcktInfo(Ipv6Pckt),

    TrafficClass = PcktInfo#ipv6PckInfo.trafficClass,
    FlowLabel = PcktInfo#ipv6PckInfo.flowLabel,
    NextHeader = PcktInfo#ipv6PckInfo.nextHeader,
    HopLimit = PcktInfo#ipv6PckInfo.hopLimit,
    SourceAddress = PcktInfo#ipv6PckInfo.sourceAddress,
    DestAddress = PcktInfo#ipv6PckInfo.destAddress,

    Map = #{},
    List = [],

    {CID, UpdateMap0, UpdatedList0} =
        encodeCid(SourceAddress, DestAddress, Map, List),

    {TF, UpdateMap1, UpdatedList1} =
        encodeTf(TrafficClass, FlowLabel, UpdateMap0, UpdatedList0),

    {NH, UpdateMap2, UpdatedList2} = encodeNh(NextHeader, UpdateMap1, UpdatedList1
        ),

    {HLIM, UpdateMap3, UpdatedList3} = encodeHlim(HopLimit, UpdateMap2,
        UpdatedList2),

    SAC = encodeSac(SourceAddress),

    {SAM, UpdateMap4, UpdatedList4} =
        encodeSam(CID, SAC, SourceAddress, UpdateMap3, UpdatedList3, RouteExist),

    M = encodeM(DestAddress),

    DAC = encodeDac(DestAddress),

    {DAM, CarrInlineMap, CarrInlineList} =
        encodeDam(CID, M, DAC, DestAddress, UpdateMap4, UpdatedList4, RouteExist),

    %CH = {TF, NH, HLIM, CID, SAC, SAM, M, DAC, DAM, CarrInlineList},

    CarrInlineBin = list_to_binary(CarrInlineList),
    % io:format("Actual carried values: ~p ~n",[CarrInlineMap]),
    case NextHeader of
        ?UDP_PN ->
            UdpPckt = getUdpData(Ipv6Pckt),
            CompressedUdpHeaderBin = compressUdpHeader(UdpPckt, []),
            io:format("Lowpan core: UdpInline: ~p ~n",[CompressedUdpHeaderBin]),
```

```
            CompressedHeader =
                <<?IPHC_DHTYPE:3, TF:2, NH:1, HLIM:2, CID:1, SAC:1, SAM:2, M:1, DAC:1, DAM:2, CarrInlineBin/binary, CompressedUdpHeaderBin/binary>>,
            {CompressedHeader, CarrInlineMap};
        _ ->
            CompressedHeader =
                <<?IPHC_DHTYPE:3, TF:2, NH:1, HLIM:2, CID:1, SAC:1, SAM:2, M:1, DAC:1, DAM:2, CarrInlineBin/binary>>,
            {CompressedHeader, CarrInlineMap}
    end.

%-------------------------------------------------------------------------------
%% @private
%% @doc Encodes the TrafficClass and Flow label fields
%% @spec encodeTf(TrafficClass, FlowLabel, CarrInlineMap, CarrInlineList) -> {integer(), map(), list()}.
%% @returns a tuple containing the compressed values and the CarrInline values
%-------------------------------------------------------------------------------
-spec encodeTf(TrafficClass, FlowLabel, CarrInlineMap, CarrInlineList) -> {term(), map(), list()} when
      TrafficClass :: integer(),
      FlowLabel :: integer(),
      CarrInlineMap :: map(),
      CarrInlineList :: list().
encodeTf(TrafficClass, FlowLabel, CarrInlineMap, CarrInlineList) ->
    <<DSCP:6, ECN:2>> = <<TrafficClass:8>>,

    case {ECN, DSCP, FlowLabel} of
        {0, 0, 0} ->
            % Traffic Class and Flow Label are elided
            {2#11, CarrInlineMap, CarrInlineList};

        {_, 0, _} ->
            % DSCP is elided
            UpdatedMap = CarrInlineMap#{"ECN" => ECN, "FlowLabel" => FlowLabel},
            Bin = <<ECN:2, 0:2, FlowLabel:20>>, % 24 bits tot (RFC 6282 - pg12)
            L = [Bin],
            UpdatedList = [CarrInlineList, L],
            {2#01, UpdatedMap, UpdatedList};

        {_, _, 0} ->
            % Flow Label is elided
            UpdatedMap = CarrInlineMap#{"TrafficClass" => TrafficClass},
            Bin = <<ECN:2, DSCP:6>>,
            L = [Bin],
            UpdatedList = [CarrInlineList, L],
            {2#10, UpdatedMap, UpdatedList};

        _ ->
            % ECN, DSCP, and Flow Label are carried inline
            UpdatedMap = CarrInlineMap#{"TrafficClass" => TrafficClass, "FlowLabel" => FlowLabel},
            Bin = <<ECN:2, DSCP:6, 0:4, FlowLabel:20>>, % 32 bits tot (RFC 6282 - pg12)
            L = [Bin],
            UpdatedList = [CarrInlineList, L],
            {2#00, UpdatedMap, UpdatedList}
    end.

%-------------------------------------------------------------------------------
%% @private
```

```
%% @doc Encodes the NextHeader field
%% @doc NextHeader specifies whether or not the next header is encoded using NHC
%% @spec encodeNh(NextHeader, CarrInlineMap, CarrInlineList)->{integer(), map(), list()}.
%% @returns a tuple containing the compressed value and the CarrInline values
%------------------------------------------------------------------------------
-spec encodeNh(NextHeader, CarrInlineMap, CarrInlineList) -> {integer(), map(), list()} when
      NextHeader :: integer(),
      CarrInlineMap :: map(),
      CarrInlineList :: list().
encodeNh(NextHeader, CarrInlineMap, CarrInlineList) when NextHeader == ?UDP_PN ->
    {1, CarrInlineMap, CarrInlineList};
encodeNh(NextHeader, CarrInlineMap, CarrInlineList) when NextHeader == ?TCP_PN ->
    Bin = <<NextHeader:8>>,
    L = [Bin],
    UpdatedList = [CarrInlineList, L],
    {0, CarrInlineMap#{"NextHeader" => ?TCP_PN}, UpdatedList};
encodeNh(NextHeader, CarrInlineMap, CarrInlineList) when NextHeader == ?ICMP_PN ->
    Bin = <<NextHeader:8>>,
    L = [Bin],
    UpdatedList = [CarrInlineList, L],
    {0, CarrInlineMap#{"NextHeader" => ?ICMP_PN}, UpdatedList};
encodeNh(NextHeader, CarrInlineMap, CarrInlineList) ->
    Bin = <<NextHeader:8>>,
    L = [Bin],
    UpdatedList = [CarrInlineList, L],
    {0, CarrInlineMap#{"NextHeader" => NextHeader}, UpdatedList}.

%------------------------------------------------------------------------------
%% @private
%% @doc Encodes the HopLimit field
%% @spec encodeHlim(HopLimit, CarrInlineMap, CarrInlineList) -> {integer(), map(), list()}.
%% @returns a tuple containing the compressed value and the CarrInline values
%------------------------------------------------------------------------------
-spec encodeHlim(HopLimit, CarrInlineMap, CarrInlineList) -> {integer(), map(), list()} when
      HopLimit :: integer(),
      CarrInlineMap :: map(),
      CarrInlineList :: list().
encodeHlim(HopLimit, CarrInlineMap, CarrInlineList) when HopLimit == 1 ->
    {2#01, CarrInlineMap, CarrInlineList};
encodeHlim(HopLimit, CarrInlineMap, CarrInlineList) when HopLimit == 64 ->
    {2#10, CarrInlineMap, CarrInlineList};
encodeHlim(HopLimit, CarrInlineMap, CarrInlineList) when HopLimit == 255 ->
    {2#11, CarrInlineMap, CarrInlineList};
encodeHlim(HopLimit, CarrInlineMap, CarrInlineList) ->
    Bin = <<HopLimit:8>>,
    L = [Bin],
    UpdatedList = CarrInlineList ++ L,
    {2#00, CarrInlineMap#{"HopLimit" => HopLimit}, UpdatedList}.

%------------------------------------------------------------------------------
%% @private
%% @doc Encodes the Context Identifier Extension field
%% @doc If this bit is 1, an 8 bit CIE field follows after the DAM field
%% @spec encodeCid(SrcAdd, DstAdd, CarrInlineMap, CarrInlineList) -> {integer(), map(), list()}.
%% @returns a tuple containing the compressed value and the CarrInline values
%------------------------------------------------------------------------------
```

```
-spec encodeCid(SrcAdd, DstAdd, CarrInlineMap, CarrInlineList) -> {integer(), map
    (), list()} when
      SrcAdd :: binary(),
      DstAdd :: binary(),
      CarrInlineMap :: map(),
      CarrInlineList :: list().
encodeCid(SrcAdd, DstAdd, CarrInlineMap, CarrInlineList) ->
    <<SrcAddPrefix:16, _/binary>> = <<SrcAdd:128>>,
    <<DstAddPrefix:16, _/binary>> = <<DstAdd:128>>,
    SrcPrefixKey = <<SrcAddPrefix:16, 0:48>>,
    DstPrefixKey = <<DstAddPrefix:16, 0:48>>,

    % check if prefix is in contextTable
    SrcContext = maps:find(SrcPrefixKey, ?Prefixt_id_table),
    DstContext = maps:find(DstPrefixKey, ?Prefixt_id_table),

    case {SrcContext, DstContext} of
        {{ok, SrcContextId}, {ok, DstContextId}} ->
            Bin = <<SrcContextId:4, DstContextId:4>>,
            L = [Bin],
            UpdatedList = CarrInlineList ++ L,
            {1, CarrInlineMap, UpdatedList};

        {error, {ok, DstContextId}} ->
            Bin = <<0:4, DstContextId:4>>,
            L = [Bin],
            UpdatedList = CarrInlineList ++ L,
            {1, CarrInlineMap, UpdatedList};

        {{ok, SrcContextId}, error} ->
            SrcContextId = someValue,
            Bin = <<SrcContextId:4, 0:4>>,
            L = [Bin],
            UpdatedList = CarrInlineList ++ L,
            {1, CarrInlineMap, UpdatedList};

        _-> {0, CarrInlineMap, CarrInlineList}
    end.

%-------------------------------------------------------------------------------
%% @private
%% @doc Encodes the Source Address Compression
%% @doc SAC specifies whether the compression is stateless or statefull
%% @spec encodeSac(SrcAdd) -> integer().
%% @returns the compressed value
%-------------------------------------------------------------------------------
-spec encodeSac(SrcAdd) -> integer() when
      SrcAdd :: binary().
encodeSac(SrcAdd) ->
    <<Prefix:16, _/binary>> = <<SrcAdd:128>>,

    case Prefix of
        ?LINK_LOCAL_PREFIX ->
            0;
        ?MULTICAST_PREFIX ->
            0;
        _ ->
            1
    end.

%-------------------------------------------------------------------------------
%% @private
```

```erlang
%% @doc Encodes for the Source Address Mode
%% @spec encodeSam(integer(), integer(), binary(), map(), list(), boolean()) -> {
    integer(), map(), list()}.
%% @returns a tuple containing the compressed value and the CarrInline values
%-------------------------------------------------------------------------------
-spec encodeSam(integer(), integer(), binary(), map(), list(), boolean()) -> {
    integer(), map(), list()}.
encodeSam(_CID, SAC, SrcAdd, CarrInlineMap, CarrInlineList, RouteExist) when SAC
    == 0 ->
    SrcAddBits = <<SrcAdd:128>>,
    <<_:112, Last16Bits:16>> = SrcAddBits,
    <<_:64, Last64Bits:64>> = SrcAddBits,

    case {SrcAddBits, RouteExist} of
        {<<?LINK_LOCAL_PREFIX:16, 0:48, _:24, 16#FFFE:16, _:24>>, _} ->
            % the address is fully elided
            {2#11, CarrInlineMap, CarrInlineList};
        {_, true} ->
            {2#11, CarrInlineMap, CarrInlineList};

        {<<?LINK_LOCAL_PREFIX:16, 0:48, 16#000000FFFE00:48, _:16>>, _} ->
            % the first 112 bits are elided, last 16 IID bits are carried in-line
            Bin = <<Last16Bits:16>>,
            L = [Bin],
            UpdatedList = [CarrInlineList, L],
            UpdatedMap = CarrInlineMap#{"SAM" => Last16Bits},
            {2#10, UpdatedMap, UpdatedList};

        {<<?LINK_LOCAL_PREFIX:16, 0:48, _:64>>, _} ->
            % the first 64 bits are elided, last 64 bits (IID) are carried in-line
            Bin = <<Last64Bits:64>>,
            L = [Bin],
            UpdatedList = [CarrInlineList, L],
            UpdatedMap = CarrInlineMap#{"SAM" => Bin},
            {2#01, UpdatedMap, UpdatedList};
        {_, _} ->
            % full address is carried in-line
            Bin = <<SrcAdd:128>>,
            L = [Bin],
            UpdatedList = [CarrInlineList, L],
            {2#00, CarrInlineMap#{"SAM" => Bin}, UpdatedList}
    end;
encodeSam(0, 1, SrcAdd, CarrInlineMap, CarrInlineList, _RouteExist) ->
    Bin = <<SrcAdd:128>>,
    L = [Bin],
    UpdatedList = [CarrInlineList, L],
    {2#00, CarrInlineMap#{"SAM" => Bin}, UpdatedList};

encodeSam(_CID, SAC, SrcAdd, CarrInlineMap, CarrInlineList, _RouteExist) when SAC
    == 1 ->
    SrcAddBits = <<SrcAdd:128>>,
    <<_:112, Last16Bits:16>> = SrcAddBits,
    <<_:64, Last64Bits:64>> = SrcAddBits,

    case SrcAddBits of
        <<_Prefix:16, _:48, _:24, 16#FFFE:16, _:24>> ->
            % the address is fully elided
            {2#11, CarrInlineMap, CarrInlineList};

        <<_Prefix:16, _:48, 16#000000FFFE00:48, _:16>> ->
            % the first 112 bits are elided, last 16 IID bits are carried in-line
            Bin = <<Last16Bits:16>>,
```

```
            L = [Bin],
            UpdatedList = [CarrInlineList, L],
            UpdatedMap = CarrInlineMap#{"SAM" => Bin},
            {2#10, UpdatedMap, UpdatedList};

        <<_Prefix:16, _:48, _:64>> ->
            % the first 64 bits are elided, last 64 bits (IID) are carried in-line
            Bin = <<Last64Bits:64>>,
            L = [Bin],
            UpdatedList = [CarrInlineList, L],
            UpdatedMap = CarrInlineMap#{"SAM" => Bin},
            {2#01, UpdatedMap, UpdatedList};

        <<0:128>> -> % The UNSPECIFIED address, ::
            {2#00, CarrInlineMap, CarrInlineList}
    end.

%-------------------------------------------------------------------------------
%% @private
%% @doc Defines the multicast compression
%% @spec encodeM(DstAdd) -> integer().
%% @returns the compressed value
%-------------------------------------------------------------------------------
-spec encodeM(DstAdd) -> integer() when
      DstAdd :: binary().
encodeM(DstAdd) ->
    <<Prefix:16, _/bitstring>> = <<DstAdd:128>>,
    case Prefix of
        ?MULTICAST_PREFIX ->
            1;
        _ ->
            0
    end.

%-------------------------------------------------------------------------------
%% @private
%% @doc encode for the Destination Address Compression
%% @spec encodeDac(DstAdd) -> integer().
%% @doc DAC specifies whether the compression is stateless or statefull
%% @returns the compressed value
%-------------------------------------------------------------------------------
-spec encodeDac(DstAdd) -> integer() when
      DstAdd :: binary().
encodeDac(DstAdd) ->
    <<Prefix:16, _/binary>> = <<DstAdd:128>>,

    case Prefix of
        ?LINK_LOCAL_PREFIX ->
            0;
        ?MULTICAST_PREFIX ->
            0;
        _ ->
            1
    end.

%-------------------------------------------------------------------------------
%% @private
%% @doc Encodes logic for the Destination Address Mode
%% @spec encodeDam(integer(), integer(), integer(), binary(), map(),
%% list(), boolean()) -> {integer(), map(), list()}.
%% @param Cid, M, DAC, DstAdd, CarrInlineMap
%% @returns a tuple containing the compressed value and the CarrInline values
```

```
%-------------------------------------------------------------------------------
-spec encodeDam(integer(), integer(), integer(), binary(), map(), list(), boolean
    ()) -> {integer(), map(), list()}.
encodeDam(0, 0, 0, DstAdd, CarrInlineMap, CarrInlineList, RouteExist) ->
    DestAddBits = <<DstAdd:128>>,
    <<_:112, Last16Bits:16>> = DestAddBits,
    <<_:64, Last64Bits:64>> = DestAddBits,

    case {DestAddBits, RouteExist} of
        {<<?LINK_LOCAL_PREFIX:16, 0:48, _:24, 16#FFFE:16, _:24>>, _} ->
            % MAC address is split into two 24-bit parts, FFFE is inserted in the
                middle
            {2#11, CarrInlineMap, CarrInlineList};
        {_, true} -> {2#11, CarrInlineMap, CarrInlineList};

        {<<?LINK_LOCAL_PREFIX:16, 0:48, 16#000000FFFE00:48, _:16>>, _} ->
            % the first 112 bits are elided, last 16 bits are in-line
            Bin = <<Last16Bits:16>>,
            L = [Bin],
            UpdatedList = [CarrInlineList, L],
            UpdatedMap = CarrInlineMap#{"DAM" => Bin},
            {2#10, UpdatedMap, UpdatedList};

        {<<?LINK_LOCAL_PREFIX:16,  0:48, _:64>>, _} ->
            % the first 64 bits are elided, last 64 bits are in-line
            Bin = <<Last64Bits:64>>,
            L = [Bin],
            UpdatedList = [CarrInlineList, L],
            UpdatedMap = CarrInlineMap#{"DAM" => Bin},
            {2#01, UpdatedMap, UpdatedList};
        {_, _} ->
            % full address is carried in-line
            Bin = <<DstAdd:128>>,
            L = [Bin],
            UpdatedList = [CarrInlineList, L],
            {2#00, CarrInlineMap#{"DAM" => Bin}, UpdatedList}
    end;
encodeDam(1, 0, 1, DstAdd, CarrInlineMap, CarrInlineList, _RouteExist) ->
    DestAddBits = <<DstAdd:128>>,
    <<_:112, Last16Bits:16>> = DestAddBits,
    <<_:64, Last64Bits:64>> = DestAddBits,

    case DestAddBits of
        <<_Prefix:16, _:48, _:24, 16#FFFE:16, _:24>> ->
            % the address is fully elided
            {2#11, CarrInlineMap, CarrInlineList};

        <<_Prefix:16, _:48, 16#000000FFFE00:48, _:16>> ->
            % the first 112 bits are elided, last 16 IID bits are carried in-line
            Bin = <<Last16Bits:16>>,
            L = [Bin],
            UpdatedList = [CarrInlineList, L],
            UpdatedMap = CarrInlineMap#{"DAM" => Bin},
            {2#10, UpdatedMap, UpdatedList};

        <<_Prefix:16, _:48, _:64>> ->
            % the first 64 bits are elided, last 64 bits (IID) are carried in-line
            Bin = <<Last64Bits:64>>,
            L = [Bin],
            UpdatedList = [CarrInlineList, L],
            UpdatedMap = CarrInlineMap#{"DAM" => Bin},
            {2#01, UpdatedMap, UpdatedList}
```

```
    end;
encodeDam(0, 0, 1, DstAdd, CarrInlineMap, CarrInlineList, _RouteExist) ->
    Bin = <<DstAdd:128>>,
    L = [Bin],
    UpdatedList = [CarrInlineList, L],
    {2#00, CarrInlineMap#{"DAM" => Bin}, UpdatedList};

encodeDam(_CID, 1, 0, DstAdd, CarrInlineMap, CarrInlineList, _RouteExist) ->
    DestAddBits = <<DstAdd:128>>,
    <<_:80, Last48Bits:48>> = DestAddBits,
    <<_:96, Last32Bits:32>> = DestAddBits,
    <<_:120, Last8Bits:8>> = DestAddBits,
    case DestAddBits of
        % ff02::00XX.
        <<?MULTICAST_PREFIX:16, 0:104, _:8>> ->
            Bin = <<Last8Bits:8>>,
            L = [Bin],
            UpdatedList = [CarrInlineList, L],
            UpdatedMap = CarrInlineMap#{"DAM" => Bin},
            {2#11, UpdatedMap, UpdatedList};

        % ffXX::00XX:XXXX.
        <<16#FF:8, _:8, 0:80, _:32>> ->
            Bin = <<Last32Bits:32>>,
            L = [Bin],
            UpdatedList = [CarrInlineList, L],
            UpdatedMap = CarrInlineMap#{"DAM" => Bin},
            {2#10, UpdatedMap, UpdatedList};

        % ffXX::00XX:XXXX:XXXX.
        <<16#FF:8, _:8, 0:64, _:48>> ->
            Bin = <<Last48Bits:48>>,
            L = [Bin],
            UpdatedList = [CarrInlineList, L],
            UpdatedMap = CarrInlineMap#{"DAM" => Bin},
            {2#01, UpdatedMap, UpdatedList};
        _ ->
            % full address is carried in-line
            Bin = <<DstAdd:128>>,
            L = [Bin],
            UpdatedList = [CarrInlineList, L],
            {2#00, CarrInlineMap#{"DAM" => Bin}, UpdatedList}
    end;
encodeDam(_CID, 1, 1, DstAdd, CarrInlineMap, CarrInlineList, _RouteExist) ->
    DestAddBits = <<DstAdd:128>>,
    <<_:80, Last48Bits:48>> = DestAddBits,
    case DestAddBits of
        <<16#FF, _:112>> ->
            Bin = <<Last48Bits:48>>,
            L = [Bin],
            UpdatedList = [CarrInlineList, L],
            UpdatedMap = CarrInlineMap#{"DAM" => Bin},
            {2#00, UpdatedMap, UpdatedList}
    end.

%-------------------------------------------------------------------------------
%
%                        Next Header compression
%
%-------------------------------------------------------------------------------

%-------------------------------------------------------------------------------
```

```
%                         UDP Packet Compression
%-------------------------------------------------------------------------------

%-------------------------------------------------------------------------------
%                   Structure of a UDP Datagram Header
%
%    0                   1                   2                   3
%    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
%   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
%   |        Source Port            |       Destination Port        |
%   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
%   |             Length            |           Checksum            |
%   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
%

-spec compressUdpHeader(UdpPckt, CarriedInline) -> binary() when
      UdpPckt :: binary(),
      CarriedInline :: list().
compressUdpHeader(UdpPckt, CarriedInline) ->
    <<SrcPort:16, DstPort:16, _Length:16, Checksum:16>> = <<UdpPckt:64>>,

    {P, CarriedInlineList} = encodeUdpPorts(SrcPort, DstPort, CarriedInline),
    {C, CarriedIn} = encodeUdpChecksum(Checksum, CarriedInlineList),

    Inline = list_to_binary(CarriedIn),

    CompressedUdpHeader = <<?UDP_DHTYPE:5, C:1, P:2, Inline/bitstring>>,
    CompressedUdpHeader.

-spec encodeUdpPorts(SrcPort, DstPort, CarriedInline) -> {integer(), list()} when
      SrcPort :: integer(),
      DstPort :: integer(),
      CarriedInline :: list().
encodeUdpPorts(SrcPort, DstPort, CarriedInline) ->
    case {<<SrcPort:16>>, <<DstPort:16>>} of
        {<<?0xf0b:12, Last4S_Bits:4>>, <<?0xf0b:12, Last4D_Bits:4>>} ->
            ToCarr = <<Last4S_Bits:4, Last4D_Bits:4>>,
            L = [ToCarr],
            CarriedInlineList = CarriedInline ++ L,
            P = 2#11,
            {P, CarriedInlineList};
        {<<?0xf0:8, Last8S_Bits:8>>, _} ->
            ToCarr = <<Last8S_Bits:8, DstPort:16>>,
            L = [ToCarr],
            CarriedInlineList = CarriedInline ++ L,
            P = 2#10,
            {P, CarriedInlineList};
        {_, <<?0xf0:8, Last8D_Bits:8>>} ->
            ToCarr = <<SrcPort:16, Last8D_Bits:8>>,
            L = [ToCarr],
            CarriedInlineList = CarriedInline ++ L,
            P = 2#01,
            {P, CarriedInlineList};
        {_, _} ->
            P = 2#00,
            ToCarr = <<SrcPort:16, DstPort:16>>,
            L = [ToCarr],
            CarriedInlineList = CarriedInline ++ L,
            {P, CarriedInlineList}
    end.

-spec encodeUdpChecksum(Checksum, CarriedInline) -> {integer(), list()} when
```

```
      Checksum :: integer(),
      CarriedInline :: list().
encodeUdpChecksum(Checksum, CarriedInline) ->
    case Checksum of
        0 ->
            {1, CarriedInline};
        %Checksum is carried inline
        _ ->
            L = [<<Checksum:16>>],
            UpdatedList = CarriedInline ++ L,
            {0, UpdatedList}
    end.

%-------------------------------------------------------------------------------
%                        ICMP Packet Compression
%-------------------------------------------------------------------------------

%-------------------------------------------------------------------------------
%                         TCP Packet Compression
%-------------------------------------------------------------------------------

%-------------------------------------------------------------------------------
%                        Packet Compression Helper
%-------------------------------------------------------------------------------

%-------------------------------------------------------------------------------
%% @doc Creates a compressed 6lowpan packet (with iphc compression) from an Ipv6
    packet
%% @spec createIphcPckt(IphcHeader, Payload) -> binary().
%-------------------------------------------------------------------------------
-spec createIphcPckt(IphcHeader, Payload) -> binary() when
      IphcHeader :: binary(),
      Payload :: binary().
createIphcPckt(IphcHeader, Payload) ->
    <<IphcHeader/binary, Payload/bitstring>>.

%-------------------------------------------------------------------------------
%% @doc Returns value field of a given Ipv6 packet
%% @spec getPcktInfo(Ipv6Pckt) -> map().
%-------------------------------------------------------------------------------
-spec getPcktInfo(Ipv6Pckt) -> map() when
      Ipv6Pckt :: binary().
getPcktInfo(Ipv6Pckt) ->
    <<Version:4, TrafficClass:8, FlowLabel:20, PayloadLength:16, NextHeader:8,
        HopLimit:8, SourceAddress:128, DestAddress:128, Data/bitstring>> =
        Ipv6Pckt,

    Payload = case NextHeader of
                ?UDP_PN ->
                        <<_UdpFields:64, Payld/bitstring>> = Data,
                        Payld;
                _ -> Data
             end,
    PckInfo =
        #ipv6PckInfo{
            version = Version,
            trafficClass = TrafficClass,
            flowLabel = FlowLabel,
            payloadLength = PayloadLength,
            nextHeader = NextHeader,
            hopLimit = HopLimit,
            sourceAddress = SourceAddress,
```

```
            destAddress = DestAddress,
            payload = Payload
        },
    PckInfo.

%------------------------------------------------------------------------------
%% @doc Returns value field of a decoded Ipv6 packet
%% @spec getDecodeIpv6PcktInfo(Ipv6Pckt) -> map().
%------------------------------------------------------------------------------
-spec getDecodeIpv6PcktInfo(Ipv6Pckt) -> map() when
      Ipv6Pckt :: binary().
getDecodeIpv6PcktInfo(Ipv6Pckt) ->
    <<TrafficClass:8, FlowLabel:24, NextHeader:8, HopLimit:8, SourceAddress:128,
        DestAddress:128, Data/bitstring>> =
        Ipv6Pckt,

    Payload = case NextHeader of
                ?UDP_PN ->
                        <<_UdpFields:64, Payld/bitstring>> = Data,
                        Payld;
                _ -> Data
             end,
    PckInfo =
        #ipv6PckInfo{
            version = 6,
            trafficClass = TrafficClass,
            flowLabel = FlowLabel,
            payloadLength = byte_size(Payload),
            nextHeader = NextHeader,
            hopLimit = HopLimit,
            sourceAddress = SourceAddress,
            destAddress = DestAddress,
            payload = Payload
        },
    PckInfo.

%------------------------------------------------------------------------------
%% @doc Returns UDP data from a given Ipv6 packet if it contains a UDP nextHeader
%% @spec getUdpData(Ipv6Pckt) -> binary().
%------------------------------------------------------------------------------
-spec getUdpData(Ipv6Pckt) -> binary() when
      Ipv6Pckt :: binary().
getUdpData(Ipv6Pckt) ->
    <<_:320, UdpPckt:64, _/binary>> = Ipv6Pckt,
    UdpPckt.

%------------------------------------------------------------------------------
%% @doc Returns the payload of a given Ipv6 packet
%% @spec getIpv6Payload(Ipv6Pckt) -> binary().
%------------------------------------------------------------------------------
-spec getIpv6Payload(Ipv6Pckt) -> binary() when
      Ipv6Pckt :: binary().
getIpv6Payload(Ipv6Pckt) ->
    <<_:192, _:128, Payload/binary>> = Ipv6Pckt,
    Payload.

%------------------------------------------------------------------------------
%% @doc Encodes an Integer value in a binary format using an appropriate amount of
     bit
%% @spec encodeInteger(I) -> binary().
%------------------------------------------------------------------------------
-spec encodeInteger(I) -> binary() when
```

```
      I :: integer().
encodeInteger(I) when I =< 255 ->
    <<I:8>>;
encodeInteger(I) when I =< 65535 ->
    <<I:16>>;
encodeInteger(I) when I =< 4294967295 ->
    <<I:32>>;
encodeInteger(I) ->
    <<I:64>>.

%-------------------------------------------------------------------------------
%
%                               Packet fragmentation
%
%-------------------------------------------------------------------------------

%-------------------------------------------------------------------------------
%% @doc Builds subsequent fragment header
%% @spec buildFragHeader(FragHeader) -> binary().
%-------------------------------------------------------------------------------
-spec buildFragHeader(FragHeader) -> binary() when
      FragHeader :: map().
buildFragHeader(FragHeader) ->
    #frag_header{
        frag_type = FragType,
        datagram_size = DatagramSize,
        datagram_tag = DatagramTag,
        datagram_offset = DatagramOffset
    } = FragHeader,
    <<FragType:5, DatagramSize:11, DatagramTag:16, DatagramOffset:8>>.

%-------------------------------------------------------------------------------
%% @doc Builds first fragment header
%% @spec buildFirstFragHeader(FragHeader) -> binary().
%-------------------------------------------------------------------------------
-spec buildFirstFragHeader(FragHeader) -> binary() when
      FragHeader :: map().
buildFirstFragHeader(FragHeader) ->
    #frag_header{
        frag_type = FragType,
        datagram_size = DatagramSize,
        datagram_tag = DatagramTag
    } = FragHeader,
    <<FragType:5, DatagramSize:11, DatagramTag:16>>.

%-------------------------------------------------------------------------------
%% @spec buildFirstFragPckt(FragType, DatagramSize, DatagramTag, CompressedHeader, Payload) -> binary().
%-------------------------------------------------------------------------------
-spec buildFirstFragPckt(integer(), integer(), integer(), binary(), binary()) -> binary().
buildFirstFragPckt(FragType, DatagramSize, DatagramTag, CompressedHeader, Payload) ->
    <<FragType:5, DatagramSize:11, DatagramTag:16, CompressedHeader/binary, Payload/bitstring>>.

%-------------------------------------------------------------------------------
%% @doc Creates a fragmented packet
%% @spec buildDatagramPckt(DtgmHeader, Payload) -> binary().
%-------------------------------------------------------------------------------
-spec buildDatagramPckt(map(), binary()) -> binary().
buildDatagramPckt(DtgmHeader, Payload) ->
```

```erlang
    TYPE = DtgmHeader#frag_header.frag_type,
    case TYPE of
        ?FRAG1_DHTYPE ->
            Header = buildFirstFragHeader(DtgmHeader),
            <<Header/binary, Payload/bitstring>>;
        ?FRAGN_DHTYPE ->
            Header = buildFragHeader(DtgmHeader),
            <<Header/binary, Payload/bitstring>>
    end.

%-------------------------------------------------------------------------------
%% @doc Checks if a packet needs to be fragmented or not and has a valid size
%% returns a list of fragments if yes, the orginal packet if not
%% @spec triggerFragmentation(binary(), integer()) -> {boolean(), list()} | {atom(), atom()}.
%-------------------------------------------------------------------------------
-spec triggerFragmentation(binary(), integer(), boolean()) -> {boolean(), list()} | {size_err, error_frag_size}.
triggerFragmentation(CompPckt, DatagramTag, RouteExist) when byte_size(CompPckt) =< ?MAX_DTG_SIZE ->
    PcktLengt = byte_size(CompPckt),

    ValidLength = PcktLengt =< ?MAX_FRAME_SIZE,
    case ValidLength of
        false ->
            io:format("The received Ipv6 packet needs fragmentation to be transmitted~n"),
            Fragments = fragmentIpv6Packet(CompPckt, DatagramTag, RouteExist),
            {true, Fragments};
        true ->
            io:format("No fragmentation needed~n"),
            {false, CompPckt}
    end;

triggerFragmentation(_CompPckt, _DatagramTag, _RouteExist) ->
    {size_err, error_frag_size}.

%-------------------------------------------------------------------------------
%% @doc Fragments a given Ipv6 packet
%% @spec fragmentIpv6Packet(binary(), integer()) -> list().
%% @returns a list of fragmented packets having this form:
%% [{FragHeader1, Fragment1}, ..., {FragHeaderN, FragmentN}]
%-------------------------------------------------------------------------------
-spec fragmentIpv6Packet(binary(), integer(), boolean()) -> list().
fragmentIpv6Packet(CompIpv6Pckt, DatagramTag, RouteExist) when is_binary(CompIpv6Pckt) ->
    Size = byte_size(CompIpv6Pckt),
    fragProcess(CompIpv6Pckt, DatagramTag, Size, 0, [], RouteExist).

%-------------------------------------------------------------------------------
%% @private
%% @doc helper function to process the received packet
%% @returns a list of fragmented packets
%% [{Header1, Fragment1}, ..., {HeaderN, FragmentN}]
%% @spec fragProcess(binary(), integer(), integer(), integer(), list()) -> list().
%% Input :
%%   Ipv6Pckt := binary
%%   Pckt size := integer
%%   DatagramTag := integer
%%   Offset := integer
%%   Accumulator : list
%-------------------------------------------------------------------------------
```

```
-spec fragProcess(binary(), integer(), integer(), integer(), list(), boolean()) -> list().
fragProcess(<<>>, _DatagramTag, _PacketLen, _Offset, Acc, _RouteExist) ->
    lists:reverse(Acc);
fragProcess(CompIpv6Pckt, DatagramTag, PacketLen, Offset, Acc, RouteExist) ->
    MaxSize = case RouteExist of
        true-> ?MAX_FRAG_SIZE_MESH;
        false -> ?MAX_FRAG_SIZE_NoMESH
    end,
    PcktSize = byte_size(CompIpv6Pckt),
    FragmentSize = min(PcktSize, MaxSize),

    <<FragPayload:FragmentSize/binary, Rest/bitstring>> = CompIpv6Pckt,

    case Offset of
        0 ->
            Header =
                buildFirstFragHeader(#frag_header{
                    frag_type = ?FRAG1_DHTYPE,
                    datagram_size = PacketLen,
                    datagram_tag = DatagramTag,
                    datagram_offset = Offset
                });
        _ ->
            Header =
                buildFragHeader(#frag_header{
                    frag_type = ?FRAGN_DHTYPE,
                    datagram_size = PacketLen,
                    datagram_tag = DatagramTag,
                    datagram_offset = Offset
                })
    end,

    fragProcess(Rest, DatagramTag, PacketLen, Offset + 1, [{Header, FragPayload} | Acc], RouteExist).

%-------------------------------------------------------------------------------
%% @doc Check if tag exist in the map, if so generate a new one and update the tag map
%% @spec checkTagUnicity(map(), integer()) -> {integer(), map()}.
%-------------------------------------------------------------------------------
-spec checkTagUnicity(map(), integer()) -> {integer(), map()}.
checkTagUnicity(Map, Tag) ->
    Exist = maps:is_key(Tag, Map),
    case Exist of
        true ->
            NewTag = rand:uniform(?MAX_TAG_VALUE),
            checkTagUnicity(Map, NewTag);
        false ->
            NewMap = maps:put(Tag, valid, Map),
            {Tag, NewMap}
    end.

%-------------------------------------------------------------------------------
%
%                                Packet Decoding
%
%-------------------------------------------------------------------------------

%-------------------------------------------------------------------------------
%% @doc decode an Ipv6 packet header commpressed according to the IPHC compression scheme
```

```
%% @spec decodeIpv6Pckt(boolean(), binary(), binary(), binary()) -> binary() | {atom(), atom()}.
%% @returns the decoded Ipv6 packet
%-------------------------------------------------------------------------------
-spec decodeIpv6Pckt(boolean(), binary(), binary(), binary()) -> binary() | {atom(), atom()}.
decodeIpv6Pckt(RouteExist, OriginatorMacAddr, CurrNodeMacAdd, CompressedPacket) ->
    <<Dispatch:3, TF:2, NH:1, HLIM:2, CID:1, SAC:1, SAM:2, M:1, DAC:1, DAM:2, Rest/bitstring>> =
        CompressedPacket,
    case Dispatch of
        ?IPHC_DHTYPE ->
            {SrcContextId, DstContextId, Rest0} = decodeCid(CID, Rest),
            {{DSCP, ECN}, FlowLabel, Rest1} = decodeTf(TF, Rest0),
            {NextHeader, Rest2} = decodeNextHeader(NH, Rest1),
            {HopLimit, Rest3} = decodeHlim(HLIM, Rest2),
            {SourceAddress, Rest4} = decodeSam(SAC, SAM, Rest3, OriginatorMacAddr, SrcContextId, RouteExist),
            {DestAddress, Payload} = decodeDam(M, DAC, DAM, Rest4, CurrNodeMacAdd, DstContextId, RouteExist),
            PayloadLength = byte_size(Payload),
            TrafficClass = DSCP bsl 2 + ECN,

            <<Header:5, Inline/bitstring>> = Payload,

            io:format("-----------------------------------------------------~n"),
            io:format("Decoded packet~n"),
            io:format("-----------------------------------------------------~n"),
            DecodedPckt =
            case Header of
                ?UDP_DHTYPE->
                    {SrcPort, DstPort, Checksum, UdpPayload} = decodeUdpPckt(Inline),
                    Length = byte_size(UdpPayload),
                    io:format("IPv6~n"),

                    io:format("Traffic class: ~p~nFlow label: ~p~nNext header: ~p~nHop limit: ~p~nSource address: ~p~nDestination address: ~p~n",
                            [TrafficClass, FlowLabel, NextHeader, HopLimit, convert(SourceAddress), convert(DestAddress)]),
                    io:format("-----------------------------------------------------~n"),
                    io:format("UDP~n"),
                    io:format("Source port: ~p~nDestination Port: ~p~nLength: ~p~nChecksum: ~p~n",[ SrcPort, DstPort, Length, Checksum]),
                    io:format("-----------------------------------------------------~n"),
                    io:format("Data: ~p~n",[UdpPayload]),
                    io:format("-----------------------------------------------------~n"),

                    <<6:4,TrafficClass,FlowLabel:20,PayloadLength:16,NextHeader:8,HopLimit:8,
                    SourceAddress/binary,DestAddress/binary, SrcPort:16, DstPort:16, Length:16, Checksum:16, Payload/bitstring>>;

                _->
                    io:format("IPv6~n"),
                    io:format("Traffic class: ~p~nFlow label: ~p~nPayload length: ~p~nNext header: ~p~nHop limit: ~p~nSource address: ~p~
```

```
                         nDestination address: ~p~nData: ~p~n", [TrafficClass,
                         FlowLabel, PayloadLength,
                                 NextHeader, HopLimit, convert(SourceAddress),
                                     convert(DestAddress), Payload]),
                     io:format("
                         ------------------------------------------------------~n"),
                     <<6:4,TrafficClass,FlowLabel:20,PayloadLength:16,NextHeader:8,
                         HopLimit:8,
                     SourceAddress/binary,DestAddress/binary, Payload/bitstring>>
             end,

             DecodedPckt;

         _-> error_decoding
     end.

%-------------------------------------------------------------------------------
%% @private
%% @doc Decode logic for the CID field
%% @spec decodeCid(integer(), binary()) -> {integer(), integer(), binary()}.
%% @returns the decoded ContextID
%-------------------------------------------------------------------------------
-spec decodeCid(integer(), binary()) -> {integer(), integer(), binary()}.
decodeCid(CID, CarriedInline) when CID == 1 ->
    <<SrcContextId:4, DstContextId:4, Rest/bitstring>> = CarriedInline,
    {SrcContextId, DstContextId, Rest};
decodeCid(CID, CarriedInline) when CID == 0 ->
    DefaultPrefix = 0,
    {DefaultPrefix, DefaultPrefix, CarriedInline}.

%-------------------------------------------------------------------------------
%% @private
%% @doc decode logic for the TF field
%% @spec decodeTf(integer(), binary()) -> {{integer(), integer()}, integer(),
    binary()}.
%% @returns the decoded TrafficClass and FlowLabel value
%-------------------------------------------------------------------------------
-spec decodeTf(integer(), binary()) -> {{integer(), integer()}, integer(), binary
    ()}.
decodeTf(TF, CarriedInline) ->
    case TF of
        2#11 ->
            ECN = 0, DSCP = 0, FL = 0,
            {{DSCP, ECN}, FL, CarriedInline};
        2#01 ->
            <<ECN:2, _rsv:2, FL:20, Rest/bitstring>> = CarriedInline,
            DSCP = 0,
            {{DSCP, ECN}, FL, Rest};
        2#10 ->
            <<ECN:2, DSCP:6, Rest/bitstring>> = CarriedInline,
            FL = 0,
            {{DSCP, ECN}, FL, Rest};
        2#00 ->
            <<ECN:2, DSCP:6, _rsv:4, FL:20, Rest/bitstring>> = CarriedInline,
            {{DSCP, ECN}, FL, Rest}
    end.

%-------------------------------------------------------------------------------
%% @private
%% @doc Decode logic for the NH field
%% @spec decodeNextHeader(integer(), binary()) -> {integer(), binary()}.
%% @returns the decoded NextHeader value
```

```erlang
%-------------------------------------------------------------------------------
-spec decodeNextHeader(integer(), binary()) -> {integer(), binary()}.
decodeNextHeader(NH, CarriedInline) when NH == 0 ->
    <<NextHeader:8, Rest/bitstring>> = CarriedInline,
    {NextHeader, Rest};
decodeNextHeader(NH, CarriedInline) when NH == 1 ->
    {?UDP_PN, CarriedInline}.

%-------------------------------------------------------------------------------
%% @private
%% @doc Decode logic for the HLim field
%% @spec decodeHlim(integer(), binary()) -> {integer(), binary()}.
%% @returns the decoded Hop Limit value
%-------------------------------------------------------------------------------
-spec decodeHlim(integer(), binary()) -> {integer(), binary()}.
decodeHlim(HLim, CarriedInline) ->
    <<HopLimit:8, Rest/bitstring>> = CarriedInline,
    case HLim of
        2#11 ->
            {255, CarriedInline};
        2#10 ->
            {64, CarriedInline};
        2#01 ->
            {1, CarriedInline};
        2#00 ->
            {HopLimit, Rest}
    end.

%-------------------------------------------------------------------------------
%% @private
%% @doc decode logic for the SAC field
%% @spec decodeSam(integer(), integer(), binary(), binary(), integer(), boolean()) -> {binary(), binary()}.
%% @returns the decoded Source Address Mode value
%-------------------------------------------------------------------------------
-spec decodeSam(integer(), integer(), binary(), binary(), integer(), boolean()) -> {binary(), binary()}.
decodeSam(SAC, SAM, CarriedInline, MacIID, _Context, RouteExist) when SAC == 0 ->
    case {SAM, RouteExist} of
        {2#11, true} ->
            SrcAdd = <<?LINK_LOCAL_PREFIX:16, 0:48, MacIID/binary>>,
            {SrcAdd, CarriedInline};
        {2#11, false} ->
            <<_:48, IID:16>> = MacIID,
            SrcAdd = <<?LINK_LOCAL_PREFIX:16, 0:48, 0:16, 16#00FF:16, 16#FE00:16, IID:16>>,
            {SrcAdd, CarriedInline};
        {2#10, _} ->
            <<Last16Bits:16, Rest/bitstring>> = CarriedInline,
            SrcAdd = <<?LINK_LOCAL_PREFIX:16, 0:48, 16#000000FFFE00:48, Last16Bits:16>>,
            {SrcAdd, Rest};
        {2#01, _} ->
            <<Last64Bits:64, Rest/bitstring>> = CarriedInline,
            SrcAdd = <<?LINK_LOCAL_PREFIX:16, 0:48, Last64Bits:64>>,
            {SrcAdd, Rest};
        {2#00, _} ->
            <<SrcAdd:128, Rest/bitstring>> = CarriedInline,
            {SrcAdd, Rest}
    end;
decodeSam(SAC, _SAM, CarriedInline, _MacIID, 0, _RouteExist) when SAC == 1 ->
    <<SrcAdd:128, Rest/bitstring>> = CarriedInline,
```

```
    {<<SrcAdd:128>>, Rest};
decodeSam(SAC, SAM, CarriedInline, MacIID, Context, _RouteExist) when SAC == 1 ->
    SrcAddrPrefix = maps:get(Context, ?Context_id_table),
    case SAM of
        2#11 ->
            <<_:48, IID:16>> = MacIID,
            SrcAdd = <<SrcAddrPrefix/binary, 0:16, 16#00FF:16, 16#FE00:16, IID
                :16>>,
            {SrcAdd, CarriedInline};
        2#10 ->
            <<Last16Bits:16, Rest/bitstring>> = CarriedInline,
            SrcAdd = <<SrcAddrPrefix/binary, 16#000000FFFE00:48, Last16Bits:16>>,
            {SrcAdd, Rest};
        2#01 ->
            <<Last64Bits:64, Rest/bitstring>> = CarriedInline,
            SrcAdd = <<SrcAddrPrefix/binary, Last64Bits:64>>,
            {SrcAdd, Rest};
        2#00 ->
            SrcAdd = <<0:128>>,
            {SrcAdd, CarriedInline}
    end.

%-------------------------------------------------------------------------------
%% @private
%% @doc Decode logic for the DAC field
%% @spec decodeDam(integer(), integer(), integer(), binary(), binary(), integer(),
     boolean()) -> {binary(), binary()}.
%% @returns the decoded Destination Address Mode value
%-------------------------------------------------------------------------------
-spec decodeDam(integer(), integer(), integer(), binary(), binary(), integer(),
    boolean()) -> {binary(), binary()}.
decodeDam(0, 0, DAM, CarriedInline, MacIID, _Context, RouteExist) ->
    case {DAM, RouteExist} of
        {2#11, true} ->
            DstAdd = <<?LINK_LOCAL_PREFIX:16, 0:48, MacIID/binary>>,
            {DstAdd, CarriedInline};
        {2#11, false} ->
            DstAdd = <<?LINK_LOCAL_PREFIX:16, 0:48, 0:24, 16#FFFE:16, 0:24>>,
            {DstAdd, CarriedInline};
        {2#10, _} ->
            <<Last16Bits:16, Rest/bitstring>> = CarriedInline,
            DstAdd = <<?LINK_LOCAL_PREFIX:16, 0:48, 16#000000FFFE00:48, Last16Bits
                :16>>,
            {DstAdd, Rest};
        {2#01, _} ->
            <<Last64Bits:64, Rest/bitstring>> = CarriedInline,
            DstAdd = <<?LINK_LOCAL_PREFIX:16, 0:48, Last64Bits:64>>,
            {DstAdd, Rest};
        {2#00, _} ->
            <<DstAdd:128, Rest/bitstring>> = CarriedInline,
            {DstAdd, Rest}
    end;
decodeDam(0, 1, _DAM, CarriedInline, _MacIID, 0, _RouteExist) ->
    <<DstAdd:128, Rest/bitstring>> = CarriedInline,
    {<<DstAdd:128>>, Rest};
decodeDam(0, 1, DAM, CarriedInline, _MacIID, Context, _RouteExist) ->
    DstAddrPrefix = maps:get(Context, ?Context_id_table),
    case DAM of
        2#11 ->
            {<<DstAddrPrefix/binary, 0:24, 16#FFFE:16, 0:24>>, CarriedInline};
        2#10 ->
            <<Last16Bits:16, Rest/bitstring>> = CarriedInline,
```

```
            DstAdd = <<DstAddrPrefix/binary, 16#000000FFFE00:48, Last16Bits:16>>,
            {DstAdd, Rest};
        2#01 ->
            <<Last64Bits:64, Rest/bitstring>> = CarriedInline,
            DstAdd = <<DstAddrPrefix/binary, Last64Bits:64>>,
            {DstAdd, Rest};
        2#00 -> {error_reserved, CarriedInline}
    end;
decodeDam(1, 0, DAM, CarriedInline, _MacIID, _Context, _RouteExist) ->
    case DAM of
        2#11 ->
            <<Last8Bits:8, Rest/bitstring>> = CarriedInline,
            DstAdd = <<?MULTICAST_PREFIX:16, 0:104, Last8Bits>>,
            {DstAdd, Rest};
        2#10 ->
            <<Last32Bits:32, Rest/bitstring>> = CarriedInline,
            DstAdd = <<?MULTICAST_PREFIX:16, 0:80, Last32Bits:32>>,
            {DstAdd, Rest};
        2#01 ->
            <<Last48Bits:48, Rest/bitstring>> = CarriedInline,
            DstAdd = <<?MULTICAST_PREFIX:16, 0:64, Last48Bits:48>>,
            {DstAdd, Rest};
        2#00 ->
            <<DstAdd:128, Rest/bitstring>> = CarriedInline,
            {DstAdd, Rest}
    end;
decodeDam(1, 1, DAM, CarriedInline, _MacIID, _Context, _RouteExist) ->
    case DAM of
        2#00 ->
            <<Last48Bits:48, Rest/bitstring>> = CarriedInline,
            DstAdd = <<16#FF:16, 0:64, Last48Bits:48>>,
            {DstAdd, Rest}
    end.

-spec decodeUdpPckt(binary()) -> {integer(), integer(), integer(), binary()}.
decodeUdpPckt(Rest) ->
    <<C:1, P:2, Inline/bitstring>> = Rest,
    {SrcPort, DstPort, Rest1} = decodePort(P, Inline),
    {Checksum, Payload} = decodeChecksum(C, Rest1),
    {SrcPort, DstPort, Checksum, Payload}.

-spec decodePort(integer(), binary()) -> {integer(), integer(), binary()}.
decodePort(P, Inline) ->
    case P of
        2#11 ->
            <<Last4S_Bits:4, Last4D_Bits:4, Rest/bitstring>> = Inline,
            SrcPort = <<?Oxf0b:12, Last4S_Bits:4>>,
            DstPort = <<?Oxf0b:12, Last4D_Bits:4>>,
            <<S:16>> = SrcPort,
            <<D:16>> = DstPort,
            {S, D, Rest};
        2#10 ->
            <<Last8S_Bits:8, DstPort:16, Rest/bitstring>> = Inline,
            SrcPort = <<?Oxf0:8, Last8S_Bits:8>>,
            <<S:16>> = SrcPort,
            {S, DstPort, Rest};
        2#01 ->
            <<SrcPort:16, Last8D_Bits:8, Rest/bitstring>> = Inline,
            DstPort = <<?Oxf0:8, Last8D_Bits:8>>,
            <<D:16>> = DstPort,
            {SrcPort, D, Rest};
        2#00 ->
```

```
            <<SrcPort:16, DstPort:16, Rest/bitstring>> = Inline,
            {SrcPort, DstPort, Rest}
    end.

-spec decodeChecksum(integer(), binary()) -> {integer(), binary()}.
decodeChecksum(C, Inline) ->
    case C of
        1 -> {0, Inline};
        0 ->
            <<Checksum:16, Rest/bitstring>> = Inline,
            {Checksum, Rest}
    end.

%-------------------------------------------------------------------------------
%                           Packet Decompression Helper
%-------------------------------------------------------------------------------

-spec convertAddrToBin(term()) -> binary().
convertAddrToBin(Address) ->
    DestAdd = case is_integer(Address) of
        true ->
            encodeInteger(Address);
        false ->
            Address
    end,
    DestAdd.

-spec tupleToBin(tuple()) -> binary().
tupleToBin(Tuple) ->
    Elements = tuple_to_list(Tuple),
    Binaries = [elementToBinary(Elem) || Elem <- Elements],
    list_to_binary(Binaries).

-spec elementToBinary(term()) -> binary().
elementToBinary(Elem) when is_integer(Elem) ->
    encodeInteger(Elem);
elementToBinary(Elem) when is_binary(Elem) ->
    Elem;
elementToBinary(Elem) when is_tuple(Elem) ->
    tupleToBin(Elem);
elementToBinary(Elem) when is_list(Elem) ->
    list_to_binary(Elem).

%-------------------------------------------------------------------------------
%
%                                 Reassembly
%
%-------------------------------------------------------------------------------

%-------------------------------------------------------------------------------
%% @spec datagramInfo(binary()) -> map().
%% @doc helper function to retrieve datagram info
%% @returns a tuple containing useful fragment info
%-------------------------------------------------------------------------------
-spec datagramInfo(binary()) -> map().
datagramInfo(Fragment) ->
    <<FragType:5, Rest/bitstring>> = Fragment,
    case FragType of
        ?FRAG1_DHTYPE ->
            <<DatagramSize:11, DatagramTag:16, Payload/bitstring>> = Rest,
            FragInfo =
                #datagramInfo{
```

```
                    fragtype = FragType,
                    datagramSize = DatagramSize,
                    datagramTag = DatagramTag,
                    datagramOffset = 0,
                    payload = Payload
                },
            FragInfo;
        ?FRAGN_DHTYPE ->
            <<DatagramSize:11, DatagramTag:16, DatagramOffset:8, Payload/bitstring>> = Rest,
            FragInfo =
                #datagramInfo{
                    fragtype = FragType,
                    datagramSize = DatagramSize,
                    datagramTag = DatagramTag,
                    datagramOffset = DatagramOffset,
                    payload = Payload
                },
            FragInfo
    end.



%-------------------------------------------------------------------------------
%% @doc Stores fragment in ETS and check if the datagram is complete
%% @spec storeFragment(atom(), term(), integer(), binary(), integer(), integer(), integer(), term()) -> {term(), map()}.
%-------------------------------------------------------------------------------
-spec storeFragment(map(), term(), integer(), binary(), integer(), integer(), integer(), term()) -> {term(), map()}.
storeFragment(DatagramMap, Key, Offset, Payload, CurrTime, Size, Tag, _From) ->
    {Result, Map} = case ets:lookup(DatagramMap, Key) of
        [] ->
            handleNewDatagram(DatagramMap, Key, Offset, Payload, CurrTime, Size, Tag);
        [{Key, OldDatagram}] ->
            handleExistingDatagram(DatagramMap, Key, Offset, Payload, CurrTime, Size, OldDatagram)
    end,

    io:format("------------------------------------------------------~n"),
    io:format("DatagramMap after update:~n"),
    printDatagramMap(DatagramMap),
    io:format("------------------------------------------------------~n"),
    {Result, Map}.

-spec handleNewDatagram(map(), term(), integer(), binary(), integer(), integer(), integer()) -> {term(), map()}.
handleNewDatagram(DatagramMap, Key, Offset, Payload, CurrTime, Size, Tag) ->
    if byte_size(Payload) == Size ->
        ReassembledPacket = reassemble(#datagram{
            tag = Tag,
            size = Size,
            cmpt = byte_size(Payload),
            fragments = #{Offset => Payload},
            timer = CurrTime
        }),
        ets:insert(DatagramMap, {Key, ReassembledPacket}),
        {complete_first_frag, ReassembledPacket};
    true ->
        NewDatagram = #datagram{
            tag = Tag,
```

```
            size = Size,
            cmpt = byte_size(Payload),
            fragments = #{Offset => Payload},
            timer = CurrTime
        },
        ets:insert(DatagramMap, {Key, NewDatagram}),
        {incomplete_first, Key}
    end.

-spec handleExistingDatagram(map(), term(), integer(), binary(), integer(), integer(), map()) -> {term(), map()}.
handleExistingDatagram(DatagramMap, Key, Offset, Payload, CurrTime, Size, OldDatagram) ->
    Fragments = OldDatagram#datagram.fragments,
    case maps:is_key(Offset, Fragments) of
        true ->
            {duplicate, OldDatagram};
        false ->
            NewFragments = maps:put(Offset, Payload, Fragments),
            NewCmpt = OldDatagram#datagram.cmpt + byte_size(Payload),
            UpdatedDatagram = OldDatagram#datagram{
                cmpt = NewCmpt,
                fragments = NewFragments,
                timer = CurrTime
            },
            ets:insert(DatagramMap, {Key, UpdatedDatagram}),
            if NewCmpt == Size ->
                {complete, UpdatedDatagram};
            true ->
                {incomplete, UpdatedDatagram}
            end
    end.

-spec printDatagramMap(map()) -> ok.
printDatagramMap(DatagramMap) ->
    List = ets:tab2list(DatagramMap),
    lists:foreach(fun({Key, Value}) -> printEntry(Key, Value) end, List).

-spec printEntry(term(), tuple()) -> ok.
printEntry(Key, {datagram, Tag, Size, Cmpt, Timer, Fragments}) ->
    io:format("~p -> {datagram, ~p, ~p, ~p,~n    #{~n", [Key, Tag, Size, Cmpt]),
    printFragments(Fragments),
    io:format("    }, ~p}~n", [Timer]).

-spec printFragments(map()) -> ok.
printFragments(Fragments) ->
    maps:fold(fun(Offset, Payload, Acc) ->
                    io:format("        ~p => ~p,~n", [Offset, Payload]),
                    Acc
              end, ok, Fragments).

%-------------------------------------------------------------------------------
%% @spec reassemble(map()) -> binary().
%% @doc Reassemble the datagram from stored fragments
%-------------------------------------------------------------------------------
-spec reassemble(map()) -> binary().
reassemble(Datagram) ->
    FragmentsMap = Datagram#datagram.fragments,
    SortedFragments =
        lists:sort([{Offset, Fragment} || {Offset, Fragment} <- maps:to_list(FragmentsMap)]),
    lists:foldl(
```

```
        fun({_Offset, Payload}, Acc) ->
            <<Acc/binary, Payload/binary>>
        end,
        <<>>,
        SortedFragments
    ).

%------------------------------------------------------------------------------
%
%                                     ROUTING
%
%------------------------------------------------------------------------------

%------------------------------------------------------------------------------
%                       Mesh Addressing Type and Header
%
%    0                   1                   2                   3
%    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
%   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
%   |1 0|V|F|HopsLft|  originator address,final destination address
%   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
%

%------------------------------------------------------------------------------
%% @doc Creates mesh header binary
%% @spec buildMeshHeader(map()) -> binary().
%------------------------------------------------------------------------------
-spec buildMeshHeader(map()) -> binary().
buildMeshHeader(MeshHeader) ->
    #mesh_header{
        v_bit = VBit,
        f_bit = FBit,
        hops_left = HopsLeft,
        originator_address = OriginatorAddress,
        final_destination_address = FinalDestinationAddress
    } = MeshHeader,
    <<?MESH_DHTYPE:2, VBit:1, FBit:1, HopsLeft:4,
                 OriginatorAddress/binary, FinalDestinationAddress/binary>>.

%------------------------------------------------------------------------------
%% @spec createNewMeshDatagram(binary(), binary(), binary()) -> binary().
%% @doc Creates new mesh header and returns new datagram
%------------------------------------------------------------------------------
-spec createNewMeshDatagram(binary(), binary(), binary()) -> binary().
createNewMeshDatagram(Datagram, SenderMacAdd, DstMacAdd) ->
    VBit =
        if
            byte_size(SenderMacAdd) =:= 8 -> 0;
            true -> 1
        end,
    FBit =
        if
            byte_size(DstMacAdd) =:= 8 -> 0;
            true -> 1
        end,

    MeshHeader =
        #mesh_header{
            v_bit = VBit,
            f_bit = FBit,
            hops_left = ?Max_Hops,
            originator_address = SenderMacAdd,
```

```erlang
            final_destination_address = DstMacAdd
        },
    BinMeshHeader = buildMeshHeader(MeshHeader),
    <<BinMeshHeader/binary, Datagram/bitstring>>.

%-----------------------------------------------------------------------------
%% @doc Creates new mesh header
%% @spec createNewMeshHeader(binary(), binary(), boolean()) -> binary().
%-----------------------------------------------------------------------------
createNewMeshHeader(SenderMacAdd, DstMacAdd, Extended_hopsleft) ->
    VBit =
        if
            byte_size(SenderMacAdd) =:= 8 -> 0;
            true -> 1
        end,
    FBit =
        if
            byte_size(DstMacAdd) =:= 8 -> 0;
            true -> 1
        end,

    case Extended_hopsleft of
        true ->
            <<?MESH_DHTYPE:2, VBit:1, FBit:1, ?DeepHopsLeft:4,
            SenderMacAdd/binary, DstMacAdd/binary, ?Max_DeepHopsLeft:8>>;
        false ->
            <<?MESH_DHTYPE:2, VBit:1, FBit:1, ?Max_Hops:4,
            SenderMacAdd/binary, DstMacAdd/binary>>
    end.

%-----------------------------------------------------------------------------
%% @doc Returns routing info in mesh header
%% @spec getMeshInfo(binary()) -> map().
%-----------------------------------------------------------------------------
-spec getMeshInfo(binary()) -> map().
getMeshInfo(Datagram) ->
    <<_:2, _V:1, _F:1, Hops_left:4, _/bitstring>> = Datagram,

    case Hops_left of
        ?DeepHopsLeft ->
            <<?MESH_DHTYPE:2, VBit:1, FBit:1, HopsLeft:4, OriginatorAddress:64,
                FinalDestinationAddress:64, DeepHopsLeft:8, Data/bitstring>> =
            Datagram;
        _ ->
            <<?MESH_DHTYPE:2, VBit:1, FBit:1, HopsLeft:4, OriginatorAddress:64,
                FinalDestinationAddress:64, Data/bitstring>> =
            Datagram,
            DeepHopsLeft = undefined
    end,
    MeshInfo =
        #meshInfo{
            v_bit = VBit,
            f_bit = FBit,
            hops_left = HopsLeft,
            originator_address = <<OriginatorAddress:64>>,
            final_destination_address = <<FinalDestinationAddress:64>>,
            deep_hops_left =  DeepHopsLeft,
            payload = Data
        },
    MeshInfo.

%-----------------------------------------------------------------------------
```

```
%% @doc Checks if datagram in mesh type, if so return true and mesh header info
%% @spec containsMeshHeader(binary()) -> {boolean(), map()} | boolean().
%------------------------------------------------------------------------------
-spec containsMeshHeader(binary()) -> {boolean(), map()} | boolean().
containsMeshHeader(Datagram) ->
    case Datagram of
        <<Dispatch:2, _/bitstring>> when Dispatch == ?MESH_DHTYPE ->
            {true, getMeshInfo(Datagram)};
        _ ->
            false
    end.

%------------------------------------------------------------------------------
%% @doc Removes mesh header if the datagram was meshed (used in put and reasssemble)
%% @spec removeMeshHeader(binary(), integer()) -> binary().
%------------------------------------------------------------------------------
-spec removeMeshHeader(binary(), integer()) -> binary().
removeMeshHeader(Datagram, HopsLeft) ->
    case Datagram of
        <<?MESH_DHTYPE:2, _/bitstring>> ->
            case HopsLeft of
                    ?DeepHopsLeft ->
                        <<?MESH_DHTYPE:2, _Header:142, Rest/bitstring>> = Datagram,
                        Rest;
                    _->
                        <<?MESH_DHTYPE:2, _Header:134, Rest/bitstring>> = Datagram,
                        Rest
            end;
        _ ->
            Datagram
    end.

%------------------------------------------------------------------------------
%% @doc Checks the next hop in the routing table and create new datagram with mesh
%% header if meshing is needed
%% @spec getNextHop(binary(), binary(), binary(), binary(), integer(),
% boolean()) -> {boolean(), binary(), map()} | {boolean(), binary(), map(), map()}.
%% returns a tuple {nexthop:boolean, binary, datagram, macHeader}
%------------------------------------------------------------------------------
-spec getNextHop(CurrNodeMacAdd, SenderMacAdd, DestMacAddress, DestAddress, SeqNum, HopsleftExtended) ->
      {boolean(), binary(), mac_header()}
      when
      CurrNodeMacAdd :: binary(),
      SenderMacAdd :: binary(),
      DestMacAddress :: binary(),
      DestAddress :: binary(),
      SeqNum :: integer(),
      HopsleftExtended :: boolean().
getNextHop(CurrNodeMacAdd, SenderMacAdd, DestMacAddress, DestAddress, SeqNum, Hopsleft_extended) ->
    case <<DestAddress:128>> of
        <<16#FF:8,_/binary>> ->
            MulticastAddr = generateMulticastAddr(<<DestAddress:128>>),
            Multicast_EU64 = generateEUI64MacAddr(MulticastAddr),
            MHdr = #mac_header{src_addr = CurrNodeMacAdd, dest_addr = Multicast_EU64},
            BroadcastHeader = createBroadcastHeader(SeqNum),
```

```erlang
            MeshHdrBin = createNewMeshHeader(SenderMacAdd, DestMacAddress, Hopsleft_extended),
            Header = <<MeshHdrBin/bitstring, BroadcastHeader/bitstring>>,
            {false, Header, MHdr};
        _->
            case routing_table:getRoute(DestMacAddress) of
                NextHopMacAddr when NextHopMacAddr =/= DestMacAddress ->
                    io:format("Next hop found: ~p~n", [NextHopMacAddr]),
                    MacHdr = #mac_header{src_addr = CurrNodeMacAdd, dest_addr = NextHopMacAddr},
                    MeshHdrBin = createNewMeshHeader(SenderMacAdd, DestMacAddress, Hopsleft_extended),
                    {true, MeshHdrBin, MacHdr};
                NextHopMacAddr when NextHopMacAddr == DestMacAddress ->
                    io:format("Direct link found ~n"),
                    MHdr = #mac_header{src_addr = CurrNodeMacAdd, dest_addr = DestMacAddress},
                    {false, <<>>, MHdr};
                _ ->
                    {false, <<>>, undefined, undefined}
            end
        end.

-spec getNextHop(binary(), binary()) -> {boolean(), binary(), map()} | {boolean(), binary(), map(), map()}.
getNextHop(CurrNodeMacAdd, DestMacAddress) ->
    case routing_table:getRoute(DestMacAddress) of
        NextHopMacAddr when NextHopMacAddr =/= DestMacAddress ->
        MacHdr = #mac_header{src_addr = CurrNodeMacAdd, dest_addr = NextHopMacAddr},
        MeshHdrBin = createNewMeshHeader(CurrNodeMacAdd, DestMacAddress, ?DeepHopsLeft),
        {true, MeshHdrBin, MacHdr};
    NextHopMacAddr when NextHopMacAddr == DestMacAddress ->
        MHdr = #mac_header{src_addr = CurrNodeMacAdd, dest_addr = DestMacAddress},
        {false, <<>>, MHdr};
    _ ->
        {false, <<>>, undefined, undefined}
    end.

-spec generateEUI64MacAddr(binary()) -> binary().
generateEUI64MacAddr(MacAddress) when byte_size(MacAddress) == ?SHORT_ADDR_LEN ->
    PanID = <<16#FFFF:16>>,
    Extended48Bit = <<PanID/binary, 0:16, MacAddress/binary>>,
    <<A:8, Rest:40>> = Extended48Bit,
    ULBSetup = A band 16#FD,
    <<First:16, Last:24>> = <<Rest:40>>,
    EUI64 = <<ULBSetup:8, First:16, 16#FF:8, 16#FE:8, Last:24>>,
    EUI64;
generateEUI64MacAddr(MacAddress) when byte_size(MacAddress) == ?EXTENDED_ADDR_LEN ->
    <<A:8, Rest:56>> = MacAddress,
    NewA = A bxor 2,
    <<NewA:8, Rest:56>>.

-spec getEUI64From48bitMac(binary()) -> binary().
getEUI64From48bitMac(MacAddress) ->
    <<First:24, Last:24>> = MacAddress,
    <<A:8, Rest:16>> = <<First:24>>,
    NewA = A bxor 2,
    EUI64 = <<NewA:8, Rest:16, 16#fffe:16, Last:24>>,
    EUI64.
```

```
-spec generateLLAddr(binary()) -> binary().
generateLLAddr(MacAddress) ->
    EUI64 = generateEUI64MacAddr(MacAddress),
    LLAdd = <<16#FE80:16, 0:48, EUI64/binary>>,
    LLAdd.

-spec getEUI64MacAddr(binary()) -> binary().
getEUI64MacAddr(Address) ->
    <<_:64, MacAddr:64/bitstring>> = <<Address:128>>,
    MacAddr.

-spec get16bitMacAddr(binary()) -> binary().
get16bitMacAddr(Address) ->
    <<_:112, MacAddr:16/bitstring>> = <<Address:128>>,
    MacAddr.


%-------------------------------------------------------------------------------
% Generates a EUI64 address from the 16bit short mac address
%-------------------------------------------------------------------------------
getEUI64FromShortMac(MacAddress)->
    PanID = <<16#FFFF:16>>,%ieee802154:get_pib_attribute(mac_pan_id),
    Extended48Bit = <<PanID/binary, 0:16, MacAddress/binary>>,
    <<A:8, Rest:40>> = Extended48Bit,
    ULBSetup = A band 16#FD, % replace 7th bit of first byte (U/L) by 0
    <<First:16, Last:24>> = <<Rest:40>>,
    EUI64 = <<ULBSetup:8, First:16, 16#FF:8, 16#FE:8, Last:24>>,
    EUI64.

%-------------------------------------------------------------------------------
% Generates a EUI64 address from the 64bit extended mac address
%-------------------------------------------------------------------------------
getEUI64FromExtendedMac(MacAddress)->
    <<A:8, Rest:56>> = MacAddress,
    NewA = A bxor 2,
    <<NewA:8, Rest:56>>.

-spec generateMulticastAddr(binary()) -> binary().
generateMulticastAddr(DestAddress) ->
    <<_:112, DST_15:8, DST_16:8>> = DestAddress,
    <<_:3, Last5Bits:5>> = <<DST_15:8>>,
    MulticastAddr = <<2#100:3, Last5Bits:5, DST_16:8>>,
    MulticastAddr.

-spec createBroadcastHeader(integer()) -> binary().
createBroadcastHeader(SeqNum) ->
   BC0_Header = <<?BC0_DHTYPE, SeqNum:8>>,
   BC0_Header.


%-------------------------------------------------------------------------------
%
%                               Utils functions
%
%-------------------------------------------------------------------------------

-spec convert(binary()) -> list().
convert(Binary) ->
    lists:flatten(
        lists:join(":",
```

```
            [io_lib:format("~2.16.0B", [B]) || <<B:8>> <= Binary]
        )
    ).

-spec generateChunks() -> binary().
generateChunks() ->
    NumChunks = 5,
    ChunkSize = 58,
    Chunks =
        lists:map(fun(N) -> generateChunk(N, ChunkSize) end, lists:seq(NumChunks,
            1, -1)),
    Result = lists:foldl(fun(A, B) -> <<A/binary, B/binary>> end, <<>>, Chunks),
    Result.

-spec generateChunks(integer()) -> binary().
generateChunks(Size) ->
    ChunkSize = 48,
    Chunks =
        lists:map(fun(N) -> generateChunk(N, ChunkSize) end, lists:seq(Size, 1,
            -1)),
    Result = lists:foldl(fun(A, B) -> <<A/binary, B/binary>> end, <<>>, Chunks),
    Result.

-spec generateChunk(integer(), integer()) -> binary().
generateChunk(N, Size) ->
    Prefix = list_to_binary(io_lib:format("chunk_~2..0B", [N])),
    PrefixSize = byte_size(Prefix),
    PaddingSize = Size - PrefixSize,
    Padding = list_to_binary(lists:duplicate(PaddingSize, $a)),
    <<Prefix/binary, Padding/binary>>.
```

A.5 Lowpan API code

```
-module(lowpan_api).

-behaviour(gen_statem).

-include("lowpan.hrl").

-export([init/1, start_link/1, start/1, stop_link/0, stop/0]).
-export([callback_mode/0]).
-export([sendPacket/1, sendPacket/2, sendUncDatagram/3, tx/3, extendedHopsleftTx/1
    ]).
-export([frameReception/0]).
-export([inputCallback/4]).
-export([idle/3]).
-export([tx_frame/3]).
-export([tx_datagram/3]).
-export([tx_packet/3]).
-export([rx_frame/3]).
-export([collect/3]).
-export([reassemble/3]).
-export([forward/3]).
-export([tx_packet_metrics/3]).

% API Functions
%% @doc Initializes the lowpan API module.
```

```
%% @spec init(map()) -> {ok, atom(), map()}.
init(Params) ->
    io:format("
        ---------------------------------------------------------------------------~
        n"),
    io:format("Initialization~n"),
    MacAdd = maps:get(node_mac_addr, Params),
    CurrNodeMacAdd = lowpan_core:generateEUI64MacAddr(MacAdd),
    io:format("Current node address: ~p~n",[CurrNodeMacAdd]),
    setup_node_info_ets(),

    RoutingTable  = maps:get(routing_table, Params),

    case routing_table:start_link(RoutingTable) of
        {ok, _Pid} ->
            io:format("~p: Routing table server successfully launched~n", [node()]
                );
        {error, Reason} ->
            io:format("~p: Failed to start routing table server: ~p~n", [node(),
                Reason]),
            exit({error, Reason})
    end,

    ieee802154_setup(CurrNodeMacAdd),

    DatagramMap = ets:new(datagram_map, [named_table, public]),

    Data = #{node_mac_addr => CurrNodeMacAdd, datagram_map => DatagramMap,
            fragment_tag => ?DEFAULT_TAG_VALUE, seqNum => ?BC_SEQNUM,
            metrics => #metrics{}, ack_req => false},

    set_nodeData_value(state_data, Data),

    io:format("~p: 6lowpan layer successfully launched~n", [node()]),
    io:format("
        ---------------------------------------------------------------------------~
        n"),
    {ok, idle, Data}.

%% @doc Starts the lowpan API process linked to the current process.
%% @spec start_link(map()) -> {ok, pid()} | {error, Reason}.
start_link(Params) ->
    gen_statem:start_link({local, ?MODULE}, ?MODULE, Params, []).


%% @doc Starts the lowpan API process.
%% @spec start(map()) -> {ok, pid()} | {error, Reason}.
start(Params) ->
    gen_statem:start({local, ?MODULE}, ?MODULE, Params, []).


%% @doc Stops the lowpan API process linked to the current process.
%% @spec stop_link() -> ok.
stop_link() ->
    gen_statem:stop(?MODULE).

%% @doc Stops the lowpan API process.
%% @spec stop() -> ok.
stop() ->
    io:format("lowpan layer stopped"),
    erpc:call(node(), routing_table, stop, []),
    gen_statem:stop(?MODULE).
```

```

%-----------------------------------------------------------------------------
%% @doc API function to send an IPv6 packet.
%% @spec sendPacket(binary()) -> ok | {error_multicast_src} | {
    error_unspecified_addr}.
%-----------------------------------------------------------------------------
sendPacket(Ipv6Pckt) ->
    io:format("Transmission request~n"),
    PcktInfo = lowpan_core:getPcktInfo(Ipv6Pckt),
    SrcAddress = PcktInfo#ipv6PckInfo.sourceAddress,
    DstAddress = PcktInfo#ipv6PckInfo.destAddress,

    case {<<SrcAddress:128>>, <<DstAddress:128>>} of
       {<<16#FF:16, _:112>>, _} ->
            io:format("Error, Source address cannot be a multicast address~n"),
            error_multicast_src;
        {_, <<0:128>>} ->
            io:format("Error, destination address cannot be the Unspecified
                address~n"),
            error_unspecified_addr;
        _ ->
            Extended_hopsleft = false,
            gen_statem:cast(?MODULE, {pckt_tx, Ipv6Pckt, PcktInfo,
                Extended_hopsleft, self()}),
            receive
                Response ->
                    Response
            end
    end.
%-----------------------------------------------------------------------------
%% @doc API function to send an IPv6 packet with performance metrics enabled.
%% @spec sendPacket(binary()) -> ok | {error_multicast_src} | {
    error_unspecified_addr}.
%-----------------------------------------------------------------------------
sendPacket(Ipv6Pckt, MetricEnabled) ->
    io:format("Transmission request~n"),
    PcktInfo = lowpan_core:getPcktInfo(Ipv6Pckt),
    SrcAddress = PcktInfo#ipv6PckInfo.sourceAddress,
    DstAddress = PcktInfo#ipv6PckInfo.destAddress,

    case {<<SrcAddress:128>>, <<DstAddress:128>>} of
       {<<16#FF:16, _:112>>, _} ->
            io:format("Error, Source address cannot be a multicast address~n"),
            error_multicast_src;
        {_, <<0:128>>} ->
            io:format("Error, destination address cannot be the Unspecified
                address~n"),
            error_unspecified_addr;
        _ ->
            Extended_hopsleft = false,
            Response = case MetricEnabled of
                            true ->
                                gen_statem:cast(?MODULE, {pckt_tx_with_metrics,
                                    Ipv6Pckt, PcktInfo, Extended_hopsleft, self()}
                                    );
                            false ->
                                gen_statem:cast(?MODULE, {pckt_tx, Ipv6Pckt,
                                    PcktInfo, Extended_hopsleft, self()})
                        end,
            receive
                Response ->
                    Response;
```

```
                {ok, NewMetrics} ->
                    {ok, RTT, SuccessRate, CompressionRatio} = handle_ack(
                        NewMetrics),
                    _MetricsResult = {RTT, SuccessRate, CompressionRatio},
                    io:format("-----------------Metrics report
                        --------------------~n"),
                    io:format("RTT: ~p ms~nSuccessRate: ~p~nCompressionRatio: ~p~n
                        ", [RTT, SuccessRate, CompressionRatio]),
                    io:format("
                        -----------------------------------------------------~n"),
                    ok;
                error_frag_size ->
                    error_frag_size
            end
    end.

%-------------------------------------------------------------------------------
%% @doc API function to send an IPv6 packet with extended hops left option enabled
    .
%% @spec extendedHopsleftTx(binary()) -> ok | {error_multicast_src} | {
    error_unspecified_addr} | {error_timeout}.
%-------------------------------------------------------------------------------
extendedHopsleftTx(Ipv6Pckt) ->
    io:format("New packet transmission ~n"),
    PcktInfo = lowpan_core:getPcktInfo(Ipv6Pckt),
    SrcAddress = PcktInfo#ipv6PckInfo.sourceAddress,
    DstAddress = PcktInfo#ipv6PckInfo.destAddress,

    case {<<SrcAddress:128>>, <<DstAddress:128>>} of
        {<<?MULTICAST_PREFIX:16, _Rest:112>>, _} ->
            io:format("Error, Source address cannot be a multicast address~n"),
            error_multicast_src;
        {_, <<0:128>>} ->
            io:format("Error, destination address cannot be the Unspecified
                address~n"),
            error_unspecified_addr;
        _ ->
            Extended_hopsleft = true,
            Response = gen_statem:cast(?MODULE, {pckt_tx, Ipv6Pckt, PcktInfo,
                Extended_hopsleft, self()}),
            receive
                error_frag_size ->
                    error_frag_size;
                Response ->
                    Response
            end
    end.

%-------------------------------------------------------------------------------
%% @doc API function to send an uncompressed IPv6 datagram.
%% @spec sendUncDatagram(binary(), term(), map()) -> ok | {error_timeout}.
%-------------------------------------------------------------------------------
sendUncDatagram(Ipv6Pckt, FrameControl, MacHeader) ->
    gen_statem:cast(?MODULE, {datagram_tx, Ipv6Pckt, FrameControl, MacHeader, self
        ()}),
    receive
        Response ->
            Response
    after 1000 ->
            io:format("Timeout~n"),
            error_timeout
    end.
```

```
%------------------------------------------------------------------------------
%% @doc API function to send a frame.
%% @spec tx(binary(), term(), map()) -> ok | error_nalp.
%------------------------------------------------------------------------------
tx(Frame, FrameControl, MacHeader) ->
    case Frame of
        <<?NALP_DHTYPE,_/bitstring>> ->
            io:format("The received frame is not a lowpan frame~n"),
            error_nalp;
        _->
            gen_statem:cast(?MODULE, {frame_tx, Frame, FrameControl, MacHeader,
                self()}),
            receive
                Response ->
                    Response
            end
    end.

%------------------------------------------------------------------------------
%% @doc API function to handle frame reception
%% @spec frameReception() -> term().
%------------------------------------------------------------------------------
frameReception() ->
    io:format("Reception mode~n"),
    gen_statem:cast(?MODULE, {frame_rx, self()}),
    receive
        {reassembled_packet, IsMeshedPckt, OriginatorMacAddr, CurrNodeMacAdd,
            ReassembledPacket} ->
            io:format("Datagram reassembled, start packet decoding ~n"),
            _DecodedPacket = lowpan_core:decodeIpv6Pckt(IsMeshedPckt,
                OriginatorMacAddr, CurrNodeMacAdd, ReassembledPacket),
            ReassembledPacket;
        dtg_discarded ->
            io:format("Datagram successfully discarded ~n"),
            dtg_discarded;
        {reassembly_timeout, DatagramMap, EntryKey} ->
            io:format("Reassembly timeout for entry ~p~n", [EntryKey]),
            ets:delete(DatagramMap, EntryKey),
            io:format("Entry deleted~n"),
            reassembly_timeout;
        error_nalp->
            error_nalp
    after ?REASSEMBLY_TIMEOUT ->
        reassembly_timeout
    end.

%------------------------------------------------------------------------------
% Input callback to handle new received frame.
%------------------------------------------------------------------------------
inputCallback(Frame, _, _, _) ->
    {FC, MH, Datagram} = Frame,
    {IsMeshedPckt, FinalDstMacAdd, MeshPckInfo} = case lowpan_core:
        containsMeshHeader(Datagram) of
            {true, MeshInfo} ->
                {true, MeshInfo#meshInfo.final_destination_address, MeshInfo};
            false ->
                {false, MH#mac_header.dest_addr, #{}}
    end,

    OriginatorAddr = case MeshPckInfo of
                        #{}-> MH#mac_header.src_addr;
```

```
                        _ -> MeshPckInfo#meshInfo.originator_address
                    end,

    StateData = get_nodeData_value(state_data),

    processFrame(IsMeshedPckt, MeshPckInfo, OriginatorAddr, FinalDstMacAdd, FC, MH, Datagram, StateData).

%-------------------------------------------------------------------------------
%% @doc Processes new frame.
%% @spec handleDatagram(boolean(), map(), binary(), binary(), term(), map(), binary(), map()) -> term().
%-------------------------------------------------------------------------------
processFrame(IsMeshedPckt, MeshPckInfo, OriginatorAddr, FinalDstMacAdd, FC, MH, Datagram, StateData) ->
    DestAdd = lowpan_core:convertAddrToBin(FinalDstMacAdd),
    #{node_mac_addr := CurrNodeMacAdd} = StateData,

    case DestAdd of
        CurrNodeMacAdd ->
            io:format("New frame received~n"),
            io:format("Originator              : ~p~n",[OriginatorAddr]),
            io:format("Final destination address: ~p~n", [DestAdd]),
            io:format("Current node address     : ~p~n", [CurrNodeMacAdd]),

            io:format("Final destination node reached, Forwarding to lowpan layer~n"),
            case IsMeshedPckt of
                true ->
                    HopsLeft = MeshPckInfo#meshInfo.hops_left,
                    Rest = lowpan_core:removeMeshHeader(Datagram,HopsLeft),
                    gen_statem:cast(?MODULE, {new_frame_rx, IsMeshedPckt, OriginatorAddr, Rest});
                false->
                    HopsLeft = 1,
                    Rest = lowpan_core:removeMeshHeader(Datagram,HopsLeft),
                    gen_statem:cast(?MODULE, {new_frame_rx, IsMeshedPckt, OriginatorAddr, Rest})

            end;
        ?BroadcastAdd ->
            {keep_state, rx_frame};
        _ ->
            io:format("New frame received~n"),
            io:format("Originator              : ~p~n",[OriginatorAddr]),
            io:format("Final destination address: ~p~n", [DestAdd]),
            io:format("Current node address     : ~p~n", [CurrNodeMacAdd]),
            io:format("The datagram needs to be meshed~n"),
            gen_statem:cast(?MODULE, {forward, Datagram, IsMeshedPckt, MeshPckInfo, FinalDstMacAdd, CurrNodeMacAdd, FC, MH})
    end.

%---------- States --------------------------------------------

%-------------------------------------------------------------------------------
%% @doc In this state the machine waits to received transmission/reception request
%% @spec idle(atom(), term(), map()) -> {next_state, atom(), map(), list()}.
%-------------------------------------------------------------------------------
idle(cast, {pckt_tx, Ipv6Pckt, PcktInfo, Extended_hopsleft, From}, Data) ->
    {next_state, tx_packet, Data#{data => {Ipv6Pckt, PcktInfo, Extended_hopsleft, From}}, [{next_event, internal, {tx_packet}}]};

```

```
idle(cast, {pckt_tx_with_metrics, Ipv6Pckt, PcktInfo, Extended_hopsleft, From},
    Data) ->
    {next_state, tx_packet_metrics, Data#{data => {Ipv6Pckt, PcktInfo,
        Extended_hopsleft, From}}, [{next_event, internal, {tx_packet_metrics}}]};

idle(cast, {frame_tx, Frame, FrameControl, MacHeader, From}, Data) ->
    {next_state, tx_frame, Data#{data => {Frame, FrameControl, MacHeader, From}},
        [{next_event, internal, {tx_frame}}]};


idle(cast, {datagram_tx, Ipv6Pckt, FrameControl, MacHeader, From}, Data) ->
    {next_state, tx_datagram, Data#{data => {Ipv6Pckt, FrameControl, MacHeader,
        From}}, [{next_event, internal, {tx_datagram}}]};

idle(cast, {frame_rx, From}, Data) ->
    {next_state, rx_frame, Data#{caller => From}, [{next_event, internal, {
        rx_frame}}]}.


%---------- Tx frame state --------------------------------

%------------------------------------------------------------------------------
%% @doc Handles the transmission of a frame.
%% @spec tx_frame(atom(), term(), map()) -> {next_state, atom(), map()}.
%------------------------------------------------------------------------------
tx_frame(internal, {tx_frame}, Data) ->
    #{data := {Frame, FrameControl, MacHeader, From}} = Data,
    Transmit = ieee802154:transmission({FrameControl, MacHeader, Frame}),
    case Transmit of
        {ok, _} ->
            io:format("Packet sent successfully~n"),
            From ! ok,
            {next_state, idle, Data};
        {error, Error} ->
            io:format("Transmission error: ~p~n", [Error]),
            From ! {error, Error},
            {next_state, idle, Data}
    end.

%---------- Tx datagram state -----------------------------

%------------------------------------------------------------------------------
%% @doc Handles the transmission of a datagram.
%% @spec tx_datagram(atom(), term(), map()) -> {next_state, atom(), map()}.
%------------------------------------------------------------------------------
tx_datagram(internal, {tx_datagram}, Data) ->
    #{data := {Ipv6Pckt, FrameControl, MacHeader, From}} = Data,
    Transmit = ieee802154:transmission({FrameControl, MacHeader, <<?IPV6_DHTYPE:8,
         Ipv6Pckt/bitstring>>}),
    case Transmit of
        {ok, _} ->
            From ! ok,
            {next_state, idle, Data};
        {error, Error} ->
            From ! {error, Error},
            {next_state, idle, Data}
    end.

%---------- Tx packet state -------------------------------

%------------------------------------------------------------------------------
%% @doc Handles the transmission of a packet.
```

```
%% @spec tx_packet(atom(), term(), map()) -> {next_state, atom(), map()}.
%-----------------------------------------------------------------------------
tx_packet(internal, {tx_packet}, Data) ->
    #{data := {Ipv6Pckt, PcktInfo, Extended_hopsleft, From},
        node_mac_addr := CurrNodeMacAdd, seqNum := SeqNum, fragment_tag := Tag} =
            Data,
    DestAddress = PcktInfo#ipv6PckInfo.destAddress,
    SrcAddress = PcktInfo#ipv6PckInfo.sourceAddress,
    Payload = PcktInfo#ipv6PckInfo.payload,
    DestMacAddress = lowpan_core:getEUI64MacAddr(DestAddress),
    SenderMacAdd = lowpan_core:getEUI64MacAddr(SrcAddress),
    io:format("Final destination: ~p~n", [DestMacAddress]),
    io:format("Searching next hop...~n"),
    {RouteExist, MeshedHdrBin, MH} = lowpan_core:getNextHop(CurrNodeMacAdd,
        SenderMacAdd, DestMacAddress, DestAddress, SeqNum+1, Extended_hopsleft),
    {CompressedHeader, _} = lowpan_core:compressIpv6Header(Ipv6Pckt, RouteExist),
    CompressedPacket = <<CompressedHeader/binary, Payload/bitstring>>,
    _CompressedPacketLen = byte_size(CompressedPacket),
    {FragReq, Fragments} = lowpan_core:triggerFragmentation(CompressedPacket, Tag,
         RouteExist),
    FC = #frame_control{ack_req = ?ENABLED,
                        frame_type = ?FTYPE_DATA,
                        src_addr_mode = ?EXTENDED,
                        dest_addr_mode = ?EXTENDED},

    case FragReq of
        true ->
            {Response, _NoAckCnt} = sendFragments(RouteExist, Fragments, 1,
                MeshedHdrBin, MH, FC, Tag, 0),
            NewTag = Tag+1 rem ?MAX_TAG_VALUE,
            From ! Response,
            {next_state, idle, Data#{fragments => Fragments, fragment_tag =>
                NewTag}};
        false ->
            {Response, _NoAckCnt} = sendFragment(RouteExist, Fragments,
                MeshedHdrBin, MH, FC, Tag),
            NewTag = Tag+1 rem ?MAX_TAG_VALUE,
            From ! Response,
            {next_state, idle, Data#{fragments => Fragments, fragment_tag =>
                NewTag}};
        size_err ->
            io:format("The datagram size exceed the authorized length~n"),
            From ! error_frag_size,
            {next_state, idle, Data}
    end.
tx_packet_metrics(internal, {tx_packet_metrics}, Data) ->
    #{data := {Ipv6Pckt, PcktInfo, Extended_hopsleft, From},
        node_mac_addr := CurrNodeMacAdd, seqNum := SeqNum, metrics := Metrics,
            fragment_tag := Tag} = Data,
    DestAddress = PcktInfo#ipv6PckInfo.destAddress,
    SrcAddress = PcktInfo#ipv6PckInfo.sourceAddress,
    Payload = PcktInfo#ipv6PckInfo.payload,
    DestMacAddress = lowpan_core:getEUI64MacAddr(DestAddress),
    SenderMacAdd = lowpan_core:getEUI64MacAddr(SrcAddress),
    PcktHeader = ipv6:getHeader(Ipv6Pckt),
    io:format("Final destination: ~p~n", [DestMacAddress]),
    io:format("Searching next hop...~n"),
    {RouteExist, MeshedHdrBin, MH} = lowpan_core:getNextHop(CurrNodeMacAdd,
        SenderMacAdd, DestMacAddress, DestAddress, SeqNum+1, Extended_hopsleft),
    {CompressedHeader, _} = lowpan_core:compressIpv6Header(Ipv6Pckt, RouteExist),
    CompressedPacket = <<CompressedHeader/binary, Payload/bitstring>>,
    CompressedPacketLen = byte_size(CompressedPacket),
```

```
    io:format("Compressed packet len: ~p bytes~n",[CompressedPacketLen]),
    {FragReq, Fragments} = lowpan_core:triggerFragmentation(CompressedPacket, Tag, RouteExist),
    FC = #frame_control{ack_req = ?ENABLED,
                        frame_type = ?FTYPE_DATA,
                        src_addr_mode = ?EXTENDED,
                        dest_addr_mode = ?EXTENDED},

    case FragReq of
        true ->
            NewTag = Tag+1 rem ?MAX_TAG_VALUE,
            StartTime = os:system_time(millisecond),

            NewData = Data#{caller => From, fragment_tag => NewTag, ack_req => true},
            set_nodeData_value(state_data, NewData),
            {ok, NoAckCnt} = sendFragments(RouteExist, Fragments, 1, MeshedHdrBin, MH, FC, Tag, 0),
            FragmentsNbr = length(Fragments),
            AckCounter = FragmentsNbr - NoAckCnt,
            NewMetrics = Metrics#metrics{fragments_nbr = FragmentsNbr, ack_counter = AckCounter, start_time = StartTime,
                                         pckt_len = byte_size(PcktHeader), compressed_pckt_len = byte_size(CompressedHeader)},
            MetricsResult = {ok, NewMetrics},
            ResetMetrics = Metrics#metrics{fragments_nbr = 0, ack_counter = 0, start_time = 0,
                                         pckt_len = 0, compressed_pckt_len = 0},

            ResetData = Data#{caller => From, ack_req => false, metrics => ResetMetrics},
            From ! MetricsResult,
            {next_state, idle, ResetData#{fragments => Fragments, fragment_tag => NewTag}};

        false ->
            NewTag = Tag+1 rem ?MAX_TAG_VALUE,
            StartTime = os:system_time(millisecond),
            NewData = Data#{caller => From, fragment_tag => NewTag, ack_req => true},
            set_nodeData_value(state_data, NewData),
            {_R, NoAckCnt}= sendFragment(RouteExist, Fragments, MeshedHdrBin, MH, FC, Tag),
            FragmentsNbr = 1,
            AckCounter = FragmentsNbr - NoAckCnt,
            NewMetrics = Metrics#metrics{fragments_nbr = FragmentsNbr, ack_counter = AckCounter, start_time = StartTime,
                                         pckt_len = byte_size(PcktHeader), compressed_pckt_len = byte_size(CompressedHeader)},
            MetricsResult = {ok, NewMetrics},
            ResetMetrics = Metrics#metrics{fragments_nbr = 0, ack_counter = 0, start_time = 0,
                                         pckt_len = 0, compressed_pckt_len = 0},
            ResetData = Data#{caller => From, ack_req => false, metrics => ResetMetrics},
            From ! MetricsResult,
            {next_state, idle, ResetData#{fragments => Fragments, fragment_tag => NewTag}};
        size_err ->
            io:format("The datagram size exceed the authorized length~n"),
```

```erlang
            From ! error_frag_size,
            {next_state, idle, Data}
    end.

%---------- Rx frame state ------------------------------------

%--------------------------------------------------------------------------------
%% @doc Handles the reception of a frame.
%% @spec rx_frame(atom(), term(), map()) -> {next_state, atom(), map()} | {keep_state, map()}.
%--------------------------------------------------------------------------------
rx_frame(internal, {rx_frame}, Data) ->
    #{caller := From} = Data,
    {keep_state, Data#{caller => From}};

rx_frame(cast, {frame_rx, _From}, Data) ->
    {keep_state, Data};

rx_frame(cast, {new_frame_rx, IsMeshedPckt, OriginatorAddr, Datagram}, Data) ->
    #{caller := From, node_mac_addr := CurrNodeMacAdd} = Data,
    case Datagram of
        <<?IPHC_DHTYPE:3, _Rest/bitstring>> ->
            io:format("Received a compressed datagram, starting reassembly~n"),
            From ! {reassembled_packet, IsMeshedPckt, OriginatorAddr, CurrNodeMacAdd, Datagram},
            {next_state, idle, Data};

        <<?IPV6_DHTYPE:8, Payload/bitstring>> ->
            io:format("Received a uncompressed IPv6 datagram, starting reassembly~n"),
            From ! {reassembled_packet, IsMeshedPckt, OriginatorAddr, CurrNodeMacAdd, Payload},
            {next_state, idle, Data};

        <<Type:5, _Rest/bitstring>> when Type =:= ?FRAG1_DHTYPE; Type =:= ?FRAGN_DHTYPE ->
            FragInfo = lowpan_core:datagramInfo(Datagram),
            Info = FragInfo#datagramInfo.datagramTag,
            NewData = Data#{additional_info => Info},
            io:format("Storing fragment~n"),
            gen_statem:cast(?MODULE, {add_fragment, IsMeshedPckt, OriginatorAddr, Datagram}),
            {keep_state, NewData}
    end;

rx_frame(cast, {add_fragment, IsMeshedPckt, OriginatorAddr, Datagram}, Data) ->
    {next_state, collect, Data#{is_meshed_pckt => IsMeshedPckt, originator_addr => OriginatorAddr, datagram => Datagram},
    [{next_event, internal, {start_collect}}]};

rx_frame(cast, {forward, Datagram, IsMeshedPckt, MeshPckInfo, FinalDstMacAdd, CurrNodeMacAdd, FC, MH}, Data) ->
    NewData = Data#{datagram => Datagram, is_meshed_pckt => IsMeshedPckt,
                    mesh_pck_info => MeshPckInfo, final_dst_mac_add => FinalDstMacAdd,
                    curr_node_mac_add => CurrNodeMacAdd, fc => FC, mh => MH},

    {next_state, forward, NewData, [{next_event, internal, {start_forward}}]}.


%---------- Rx new frame state ------------------------------

```

```erlang

%---------- Collect state ---------------------------------

%---------------------------------------------------------------------------
%% @doc Handles the collection of fragments.
%% @spec collect(atom(), term(), map()) -> {next_state, atom(), map()} | {keep_state, map()}.
%---------------------------------------------------------------------------
collect(internal, {start_collect}, Data) ->
    #{is_meshed_pckt := IsMeshedPckt, originator_addr := OriginatorAddr, datagram := Datagram,
    datagram_map := DatagramMap, caller := From, node_mac_addr := CurrNodeMacAdd} = Data,

    DtgInfo = lowpan_core:datagramInfo(Datagram),

    Size = DtgInfo#datagramInfo.datagramSize,
    Tag = DtgInfo#datagramInfo.datagramTag,
    Offset = DtgInfo#datagramInfo.datagramOffset,
    Payload = DtgInfo#datagramInfo.payload,

    Key = {OriginatorAddr, Tag},
    CurrTime = os:system_time(second),
    case lowpan_core:storeFragment(DatagramMap, Key, Offset, Payload, CurrTime, Size, Tag, From) of
        {complete_first_frag, ReassembledPacket} ->
            io:format("Complete for pckt ~p~n", [Key]),
            From ! {reassembled_packet, IsMeshedPckt, OriginatorAddr, CurrNodeMacAdd, ReassembledPacket},
            {next_state, idle, Data};

        {complete, UpdatedDatagram} ->
            gen_statem:cast(?MODULE, {complete, IsMeshedPckt, OriginatorAddr, Key, UpdatedDatagram}),
            NewData = Data#{key => Key},
            {keep_state, NewData};

        {duplicate, _} ->
            io:format("Duplicate frame detected~n"),
            NewData = Data#{key => Key},
            {next_state, rx_frame, NewData};

         {incomplete_first, EntryKey} ->
            io:format("Incomplete first datagram, waiting for other fragments ~n"),
            erlang:send_after(?REASSEMBLY_TIMEOUT, From, {reassembly_timeout, DatagramMap, EntryKey}),
            NewData = Data#{key => Key},
            {next_state, rx_frame, NewData};

        {incomplete, _} ->
            io:format("Incomplete datagram, waiting for other fragments ~n"),
            NewData = Data#{key => Key},
            {next_state, rx_frame, NewData}
    end;

collect(cast, {complete, IsMeshedPckt, OriginatorAddr, Key, UpdatedDatagram}, Data) ->
    NewData =  Data#{is_meshed_pckt => IsMeshedPckt, originator_addr => OriginatorAddr,
                    key => Key, updated_datagram => UpdatedDatagram},
```

```
    {next_state, reassemble, NewData, [{next_event, internal, {start_reassemble}}]
        }.

%---------- Reassembly state ------------------------------

%--------------------------------------------------------------------------------
%% @doc Handles the reassembly of fragments.
%% @spec reassemble(atom(), term(), map()) -> {next_state, atom(), map()}.
%--------------------------------------------------------------------------------
reassemble(internal, {start_reassemble}, Data) ->
    %io:format("Data: ~p~n", [Data]),
    #{datagram_map := DatagramMap, caller := From, additional_info:=Info,
        node_mac_addr := CurrNodeMacAdd,
       is_meshed_pckt := IsMeshedPckt, originator_addr := OriginatorAddr,
       key := Key, updated_datagram := UpdatedDatagram} = Data,

    ReassembledPacket = lowpan_core:reassemble(UpdatedDatagram),
    io:format("Complete for pckt ~p~n", [Key]),
    ets:delete(DatagramMap, Key),
    case Info of
        ?INFO_ON ->
            From ! {additional_info, Info, ReassembledPacket};
        _ ->
            From ! {reassembled_packet, IsMeshedPckt, OriginatorAddr,
                CurrNodeMacAdd, ReassembledPacket}
    end,
    {next_state, idle, Data}.

%---------- Forward state ---------------------------------

%--------------------------------------------------------------------------------
%% @doc Handles the forwarding of datagrams.
%% @spec forward(atom(), term(), map()) -> {next_state, atom(), map()}.
%--------------------------------------------------------------------------------
forward(internal, {start_forward}, Data) ->
    #{datagram := Datagram, is_meshed_pckt := IsMeshedPckt,
                    mesh_pck_info := MeshPckInfo, final_dst_mac_add :=
                        FinalDstMacAdd,
                    curr_node_mac_add := CurrNodeMacAdd, fc := FC, mh := MH} =
                        Data,
    NewDatagram =
        case IsMeshedPckt of
            true ->
                update_datagram(MeshPckInfo, Datagram, Data);
            false ->
                SenderMacAdd = MH#mac_header.src_addr,
                lowpan_core:createNewMeshDatagram(Datagram, SenderMacAdd,
                    FinalDstMacAdd)
        end,
    case NewDatagram of
        {discard, _} ->
            {next_state, rx_frame, Data};
        _ ->
            DestMacAddress = lowpan_core:convertAddrToBin(FinalDstMacAdd),
            io:format("Searching next hop in the routing table...~n"),
            NextHopAddr = routing_table:getRoute(DestMacAddress),

            case NextHopAddr of
                DestMacAddress ->
                    io:format("Direct link found~nForwarding to node: ~p~n", [
                        NextHopAddr]);
                _ ->
```

```
                    io:format("Next hop found~nForwarding to node: ~p~n", [NextHopAddr])
            end,
            NewMH = MH#mac_header{src_addr = CurrNodeMacAdd, dest_addr = NextHopAddr},
            io:format("-------------------------------------------------------~n"),
            forward_datagram(NewDatagram, FC, NewMH, Data)
    end.

%---------- Utility functions -----------------------------------

%-------------------------------------------------------------------------------
%% @doc Sends a fragment
%% @spec sendFragment(boolean(), binary(), binary(), map(), term(), integer()) -> {ok, integer()} | {Error, integer()}.
%-------------------------------------------------------------------------------
sendFragment(RouteExist, CompressedPacket, MeshedHdrBin, MH, FC, Tag) ->
    Pckt = case RouteExist of
                true ->
                    <<MeshedHdrBin/binary, CompressedPacket/bitstring>>;
                false ->
                    CompressedPacket
            end,
    MacHeader = MH#mac_header{seqnum = Tag},
    case ieee802154:transmission({FC, MacHeader, Pckt}) of
        {ok, _} ->
            io:format("~p-byte packet successfully sent~n", [ byte_size(Pckt)]),
            {ok, 0};
        {error, Error} ->
            io:format("Transmission error: ~p~n", [Error]),
            NoAck = 1,
            {Error, NoAck}
    end.

%-------------------------------------------------------------------------------
%% @doc Sends list of fragments
%% @spec sendFragments(boolean(), list(), integer(), binary(), map(), term(), integer(), integer()) -> {ok, integer()}.
%-------------------------------------------------------------------------------
sendFragments(RouteExist, [{FragHeader, FragPayload} | Rest], PcktCounter, MeshedHdrBin, MH, FC, Tag, NoAckCnt) ->
    Pckt = case RouteExist of
                true ->
                    <<MeshedHdrBin/binary, FragHeader/binary, FragPayload/bitstring>>;
                false ->
                    <<FragHeader/binary, FragPayload/bitstring>>
            end,
    MacHeader = MH#mac_header{seqnum = Tag+PcktCounter},
    case ieee802154:transmission({FC, MacHeader, Pckt}) of
        {ok, _} ->
            io:format("~pth fragment: ~p bytes sent~n", [PcktCounter, byte_size(Pckt)]),
            sendFragments(RouteExist, Rest, PcktCounter + 1, MeshedHdrBin, MacHeader, FC, Tag, NoAckCnt);
        {error, Error} ->
            io:format("Error during transmission of fragment ~p: ~p~n", [PcktCounter, Error]),
            sendFragments(RouteExist, Rest, PcktCounter+1, MeshedHdrBin, MacHeader, FC, Tag, NoAckCnt + 1)
    end;
```

```
sendFragments(_RouteExist, [], _PcktCounter, _MeshedHdrBin, _MH, _FC, _Tag, NoAckCnt) ->
    case NoAckCnt of
        0 ->
            io:format("Packet successfully sent~n");
        _->
            io:format("Issue during transmission~n")
    end,
    {ok, NoAckCnt}.

%-------------------------------------------------------------------------------
%% @doc Updates the datagram with new mesh header information.
%% @spec update_datagram(map(), binary(), map()) -> binary() | {discard, term()}.
%-------------------------------------------------------------------------------
update_datagram(MeshInfo, Datagram, Data) ->
    HopsLeft = MeshInfo#meshInfo.hops_left,

    {Is_Extended_hopsleft, HopLft} =
        case HopsLeft of
                ?DeepHopsLeft ->
                    HopsLft = MeshInfo#meshInfo.deep_hops_left-1,
                    {true, HopsLft};
                 _ -> HopsLft = HopsLeft-1,
                    {false, HopsLft}
        end,

    case {Is_Extended_hopsleft, HopLft}  of
        {_, 0} ->
            {discard, discard_datagram(Datagram, Data)};
        {false, _} ->
            Payload = MeshInfo#meshInfo.payload,
            MeshHeader =
                #mesh_header{v_bit = MeshInfo#meshInfo.v_bit,
                             f_bit = MeshInfo#meshInfo.f_bit,
                             hops_left = HopsLft,
                             originator_address = MeshInfo#meshInfo.originator_address,
                             final_destination_address =  MeshInfo#meshInfo.final_destination_address},

            BinMeshHeader = lowpan_core:buildMeshHeader(MeshHeader),
            <<BinMeshHeader/binary, Payload/bitstring>>;
        {true, _} ->
            Payload = MeshInfo#meshInfo.payload,
            VBit = MeshInfo#meshInfo.v_bit,
            FBit = MeshInfo#meshInfo.f_bit,
            OriginatorAddress = MeshInfo#meshInfo.originator_address,
            FinalDestinationAddress =  MeshInfo#meshInfo.final_destination_address,

            BinMeshHeader = <<?MESH_DHTYPE:2, VBit:1, FBit:1, ?DeepHopsLeft:4,
                              OriginatorAddress/binary, FinalDestinationAddress/binary, HopLft:8>>,
            <<BinMeshHeader/binary, Payload/bitstring>>
    end.

%-------------------------------------------------------------------------------
%% @doc Discards the datagram when hop count reaches zero.
%% @spec discard_datagram(binary(), map()) -> {next_state, atom(), map()}.
%-------------------------------------------------------------------------------
discard_datagram(_, Data = #{caller := From})->
    io:format("Hop left value: 0, discarding the datagram~n"),
```

```
    From ! dtg_discarded,
    {next_state, rx_frame, Data}.

%-----------------------------------------------------------------------------
%% @doc Forwards a datagram to the next hop.
%% @spec forward_datagram(binary(), term(), map(), map()) -> {next_state, atom(),
    map()}.
%-----------------------------------------------------------------------------
forward_datagram(Frame, FrameControl, MacHeader, Data = #{caller := From}) ->
    case Frame of
        <<?NALP_DHTYPE,_/bitstring>> ->
            io:format("The received frame is not a lowpan frame~n"),
            From ! error_nalp;
        _->
            Transmit = ieee802154:transmission({FrameControl, MacHeader, Frame}),
            case Transmit of
                {ok, _} ->
                    io:format("Packet sent successfully~n");
                {error, Error} ->
                    io:format("Transmission error: ~p~n", [Error])
            end
    end,
    io:format("-------------------------------------------------------~n"),
    {next_state, rx_frame, Data}.

handle_ack(Metrics) ->
    TotalFragments = Metrics#metrics.fragments_nbr,
    AckCounter = Metrics#metrics.ack_counter,
    EndTime = os:system_time(millisecond),

    RTT = EndTime - Metrics#metrics.start_time,
    SuccessRate = AckCounter / TotalFragments,
    _LossRate = 1 - SuccessRate,

    OrigPcktLen = Metrics#metrics.pckt_len,
    CompPcktLen = Metrics#metrics.compressed_pckt_len,
    CompressionRatio = (CompPcktLen/OrigPcktLen),
    {ok, RTT, SuccessRate, CompressionRatio}.

callback_mode() ->
    [state_functions].
%-----------------------------------------------------------------------------
%% @doc Sets up ETS table for node information.
%% @spec setup_node_info_ets() -> atom().
%-----------------------------------------------------------------------------
setup_node_info_ets() ->
    ets:new(nodeData, [named_table, public, {keypos, 1}]).

%-----------------------------------------------------------------------------
%% @doc Sets a value in the node data ETS table.
%% @spec set_nodeData_value(term(), term()) -> ok.
%-----------------------------------------------------------------------------
set_nodeData_value(Key, Value) ->
    ets:insert(nodeData, {Key, Value}).

%-----------------------------------------------------------------------------
%% @doc Retrieves a value from the node data ETS table.
%% @spec get_nodeData_value(term()) -> term() | undefined.
%-----------------------------------------------------------------------------
get_nodeData_value(Key) ->
    case ets:lookup(nodeData, Key) of
        [] ->
```

```
            undefined;
        [{_, Value}] ->
            Value
    end.

%-----------------------------------------------------------------------------
%% @doc Sets up the IEEE 802.15.4 layer.
%% @spec ieee802154_setup(binary()) -> ok.
%-----------------------------------------------------------------------------
ieee802154_setup(MacAddr)->
    ieee802154:start(#ieee_parameters{
        phy_layer = mock_phy_network, % uncomment when testing
        duty_cycle = duty_cycle_non_beacon,
        input_callback = fun lowpan_api:inputCallback/4
    }),

    case application:get_env(robot, pan_id) of
        {ok, PanId} ->
            ieee802154:set_pib_attribute(mac_pan_id, PanId);
        _ ->
            ok
    end,

    case byte_size(MacAddr) of
        ?EXTENDED_ADDR_LEN -> ieee802154:set_pib_attribute(mac_extended_address,
            MacAddr);
        ?SHORT_ADDR_LEN -> ieee802154:set_pib_attribute(mac_short_address, MacAddr
            )
    end,

    ieee802154:rx_on(),
    io:format("~p: IEEE 802.15.4 layer successfully launched ~n",[node()]).
```

A.6 Routing table code

```
-module(routing_table).

-behaviour(gen_server).

%%% API
-export([
    start_link/1,
    stop/0,
    addRoute/2,
    deleteRoute/1,
    getRoute/1,
    updateRoute/2,
    resetRouting_table/0
]).

%%% gen_server callbacks
-export([init/1, handle_call/3, handle_cast/2, terminate/2, code_change/3]).

%%% API functions

start_link(RoutingTable) ->
    gen_server:start_link({local, ?MODULE}, ?MODULE, RoutingTable, []).
```

```

stop() ->
    gen_server:stop(?MODULE).

addRoute(DestAddr, NextHAddr) ->
    gen_server:call(?MODULE, {add_route, DestAddr, NextHAddr}).

deleteRoute(DestAddr) ->
    gen_server:call(?MODULE, {delete_route, DestAddr}).

getRoute(DestAddr) ->
    gen_server:call(?MODULE, {get_route, DestAddr}).

updateRoute(DestAddr, NextHAddr) ->
    gen_server:call(?MODULE, {update_route, DestAddr, NextHAddr}).

resetRouting_table() ->
    gen_server:call(?MODULE, reset).

%%% gen_server callbacks
init(RoutingTable) ->
    {ok, RoutingTable}.


handle_call({add_route, DestAddr, NextHAddr}, _From, RoutingTable) ->
    NewTable = maps:put(DestAddr, NextHAddr, RoutingTable),
    {reply, ok, NewTable};

handle_call({delete_route, DestAddr}, _From, RoutingTable) ->
    NewTable = maps:remove(DestAddr, RoutingTable),
    {reply, ok, NewTable};

handle_call({get_route, DestAddr}, _From, RoutingTable) ->
    NextHAddr = maps:get(DestAddr, RoutingTable, undefined),
    {reply, NextHAddr, RoutingTable};

handle_call({update_route, DestAddr, NextHAddr}, _From, RoutingTable) ->
    NewTable = maps:put(DestAddr, NextHAddr, RoutingTable),
    {reply, ok, NewTable};

handle_call(reset, _From, _MapState) ->
    {reply, ok, #{}}.

handle_cast(_, State) ->
    {noreply, State}.

terminate(_Reason, _State) ->
    ok.

code_change(_OldVsn, State, _Extra) ->
    {ok, State}.
```

A.7 Lowpan ipv6 code

```
-module(ipv6).
-export([buildIpv6UdpPacket/3, buildIpv6Header/1, buildUdpHeader/1, getHeader/1]).
-export([buildIpv6Packet/2]).
```

```erlang

-record(ipv6_header, {
    version = 2#0110, % 4-bit (version 6)
    traffic_class,  % 8-bit
    flow_label,  % 20-bit
    payload_length, % 16-bit
    next_header,  % 8-bit
    hop_limit,  % 8-bit
    source_address, % 128-bit
    destination_address % 128-bit
}).

-record(udp_header, {
    source_port, % 16-bit identifies the sender's port
    destination_port,  % 16-bit identifies the receiver's port and is required
    length,  % 16-bit indicates the length in bytes of the UDP datagram
    checksum % 16-bit may be used for error-checking of the header and data
}).

%-------------------------------------------------------------------------------
%% @doc Returns an IPv6 header in binary format
%% @spec build_ipv6_header(#ipv6_header{}) -> binary().
%-------------------------------------------------------------------------------
-spec buildIpv6Header(#ipv6_header{}) -> binary().
buildIpv6Header(IPv6Header) ->
    #ipv6_header{
        version =  _Version,
        traffic_class = Traffic_class,
        flow_label = Flow_label,
        payload_length = Payload_length,
        next_header = Next_header,
        hop_limit = Hop_limit,
        source_address = SourceAddr,
        destination_address = DestAddr
    } = IPv6Header,

    <<6:4,Traffic_class:8,Flow_label:20,Payload_length:16,Next_header:8,Hop_limit
        :8,SourceAddr/binary,DestAddr/binary>>.

%-------------------------------------------------------------------------------
%% @doc Extracts the IPv6 header from a packet
%% @spec get_header(binary()) -> binary().
%-------------------------------------------------------------------------------
-spec getHeader(binary()) -> binary().
getHeader(Ipv6Pckt) ->
    <<Header:320, _/bitstring>> = Ipv6Pckt,
    <<Header:320>>.

%-------------------------------------------------------------------------------
%% @doc Returns a UDP header in binary format
%% @spec build_udp_header(#udp_header{}) -> binary().
%-------------------------------------------------------------------------------
-spec buildUdpHeader(#udp_header{}) -> binary().
buildUdpHeader(UdpHeader) ->
    #udp_header{
        source_port =  SourcePort,
        destination_port = DestinationPort,
        length = Length,
        checksum = Checksum
    } = UdpHeader,

    <<SourcePort:16,DestinationPort:16,Length:16,Checksum:16>>.
```

```

%-----------------------------------------------------------------------------
%% @doc Builds an IPv6 packet with the given header and payload
%% @spec buildIpv6Packet(#ipv6_header{}, binary()) -> binary().
%-----------------------------------------------------------------------------
-spec buildIpv6Packet(#ipv6_header{}, binary()) -> binary().
buildIpv6Packet(IPv6Header, Payload) ->
    Header = buildIpv6Header(IPv6Header),
    IPv6Packet = <<Header/binary, Payload/bitstring>>,
    IPv6Packet.

%-----------------------------------------------------------------------------
%% @doc Builds an IPv6 packet with the given IPv6 header, UDP header, and payload
%% @spec build_ipv6_udp_packet(#ipv6_header{}, #udp_header{}, binary()) -> binary
    ().
%-----------------------------------------------------------------------------
-spec buildIpv6UdpPacket(#ipv6_header{}, #udp_header{}, binary()) -> binary().
buildIpv6UdpPacket(IPv6Header, UdpHeader, Payload) ->
    IpHeader = buildIpv6Header(IPv6Header),
    UdpH = buildUdpHeader(UdpHeader),
    IPv6Packet = <<IpHeader/binary, UdpH/binary, Payload/bitstring>>,
    IPv6Packet.
```

A.8 Utils file for testing code

```
-include("lowpan.hrl").

%-----------------------------------------------------------------------------
% Common value for testing purpose
%-----------------------------------------------------------------------------

-define(Payload, <<"Hello world this is an ipv6 packet for testing purpose">>).
-define(BigPayload, lowpan_core:generateChunks()).
-define(PayloadLength, byte_size(?Payload)).

-define(Node1Address, lowpan_core:generateLLAddr(?Node1MacAddress)). % generates a
     link local address based on the mac address
-define(Node2Address, lowpan_core:generateLLAddr(?Node2MacAddress)).
-define(Node3Address, lowpan_core:generateLLAddr(?Node3MacAddress)).
-define(Node4Address, lowpan_core:generateLLAddr(?Node4MacAddress)).
-define(Node5Address, lowpan_core:generateLLAddr(?Node5MacAddress)).

-define(IPv6Header, #ipv6_header{
    version = 6,
    traffic_class = 0,
    flow_label = 0,
    payload_length = ?PayloadLength,
    next_header = 12,
    hop_limit = 64,
    source_address = ?Node1Address,
    destination_address = ?Node2Address
}).

-define(IPv6Header3, #ipv6_header{
    version = 6,
    traffic_class = 0,
    flow_label = 0,
```

```
    payload_length = ?PayloadLength,
    next_header = 12,
    hop_limit = 64,
    source_address = ?Node1Address,
    destination_address = ?Node3Address
}).

-define(IPv6Header4, #ipv6_header{
    version = 6,
    traffic_class = 0,
    flow_label = 0,
    payload_length = ?PayloadLength,
    next_header = 12,
    hop_limit = 64,
    source_address = ?Node1Address,
    destination_address = ?Node4Address
}).

-define(IPv6Header5, #ipv6_header{
    version = 6,
    traffic_class = 0,
    flow_label = 0,
    payload_length = ?PayloadLength,
    next_header = 12,
    hop_limit = 64,
    source_address = ?Node1Address,
    destination_address = ?Node5Address
}).

-define(FrameControl, #frame_control{
    frame_type = ?FTYPE_DATA,
    src_addr_mode = ?EXTENDED,
    dest_addr_mode = ?EXTENDED
}).
-define(Ipv6Pckt, ipv6:buildIpv6Packet(?IPv6Header, ?Payload)).
-define(MacHeader, #mac_header{src_addr = ?Node1MacAddress, dest_addr = ?Node2MacAddress}).


%-------------------------------------------------------------------------------
% multiple hop Routing tables
%-------------------------------------------------------------------------------

-define(Node1_multiple_hop_routing_table,
        #{?node4_addr => ?node2_addr}).

-define(Node2_multiple_hop_routing_table,
        #{?node4_addr => ?node3_addr}).

-define(Node3_multiple_hop_routing_table,
         #{?node4_addr => ?node4_addr}).

-define(Node4_multiple_hop_routing_table,
         #{?node4_addr => ?node4_addr,
          ?node3_addr => ?node3_addr}).
```

A.9 Functional testing code

```erlang
-module(lowpan_test_SUITE).

-include("../src/utils.hrl").

-export([all/0, init_per_testcase/1, end_per_testcase/1]).
-export([
    pkt_encapsulation_test/1, fragmentation_test/1, datagram_info_test/1,
    reassemble_fragments_list_test/1, reassemble_single_fragments_test/1,
    reassemble_full_ipv6_pckt_test/1, compress_header_example1_test/1,
    compress_header_example2_test/1, link_local_addr_pckt_comp/1,
    multicast_addr_pckt_comp/1, global_context_pckt_comp1/1, udp_nh_pckt_comp/1,
    tcp_nh_pckt_comp/1, icmp_nh_pckt_comp/1, unc_ipv6/1, iphc_pckt_16bit_addr/1,
    iphc_pckt_64bit_addr/1, msh_pckt/1, extended_EUI64_from_64mac/1,
        extended_EUI64_from_48mac/1,
    extended_EUI64_from_16mac/1, check_tag_unicity/1, link_local_from_16mac/1,
        multicast_addr_validity/1,
    broadcast_pckt/1
]).
% -export([cooja_example3/1]).
% -export([cooja_example2/1]).
% -export([cooja_example1/1]).


all() ->
    [
        pkt_encapsulation_test,
        datagram_info_test,
        reassemble_fragments_list_test,
        reassemble_single_fragments_test,
        reassemble_full_ipv6_pckt_test,
        compress_header_example1_test,
        compress_header_example2_test,
        link_local_addr_pckt_comp,
        multicast_addr_pckt_comp,
        global_context_pckt_comp1,
        udp_nh_pckt_comp,
        tcp_nh_pckt_comp,
        icmp_nh_pckt_comp,
        unc_ipv6,
        iphc_pckt_64bit_addr,
        iphc_pckt_16bit_addr,
        msh_pckt, extended_EUI64_from_64mac,extended_EUI64_from_48mac,
        extended_EUI64_from_16mac, check_tag_unicity, link_local_from_16mac,
        multicast_addr_validity, broadcast_pckt
        %cooja_example1, cooja_example2, cooja_example3
    ].

init_per_testcase(Config) ->
    Config.

end_per_testcase(_Config) ->
    ok.

%-------------------------------------------------------------------------------
%                         6LoWPAN Packet Encapsulation
%-------------------------------------------------------------------------------

pkt_encapsulation_test(_Config) ->
    Payload = <<"This is an Ipv6 pckt">>,
    IPv6Header =
        #ipv6_header{
```

```
            version = 6,
            traffic_class = 0,
            flow_label = 0,
            payload_length = byte_size(Payload),
            next_header = 17,
            hop_limit = 64,
            source_address = <<1>>,
            destination_address = <<2>>
        },
    IPv6Packet = ipv6:buildIpv6Packet(IPv6Header, Payload),
    DhTypebinary = <<?IPV6_DHTYPE:8, 0:16>>,
    ToCheck = <<DhTypebinary/binary, IPv6Packet/binary>>,
    ToCheck = lowpan_core:pktEncapsulation(IPv6Header, Payload),
    ok.


unc_ipv6(_Config) ->
    Ipv6Pckt = ipv6:buildIpv6Packet(?IPv6Header, ?Payload),

    Expected = <<?IPV6_DHTYPE:8, Ipv6Pckt/bitstring>>,
    Expected = lowpan_core:getUncIpv6(Ipv6Pckt).

iphc_pckt_16bit_addr(_Config) ->
    Node1Addr = lowpan_core:generateLLAddr(<<16#0001:16>>),
    Node2Addr = lowpan_core:generateLLAddr(<<16#0002:16>>),
    IPv6Header =
        #ipv6_header{
            version = 6,
            traffic_class = 0,
            flow_label = 0,
            payload_length = byte_size(?Payload),
            next_header = 12,
            hop_limit = 64,
            source_address = Node1Addr,
            destination_address = Node2Addr
        },
    Ipv6Pckt = ipv6:buildIpv6Packet(IPv6Header,?Payload),

    InlineData = <<12:8>>,
    ExpectedHeader =
        <<?IPHC_DHTYPE:3, 3:2, 0:1, 2:2, 0:1, 0:1, 3:2, 0:1, 0:1, 3:2, InlineData/
            binary>>,

    % Create the IPHC packet
    {IPHC, _} = lowpan_core:compressIpv6Header(Ipv6Pckt, false),
    io:format("IPHC: ~p~n", [IPHC]),
    io:format("ExpectedHeader: ~p~n", [ExpectedHeader]),
    IPHC = ExpectedHeader.

iphc_pckt_64bit_addr(_Config) ->
    InlineData = <<12:8, (?node1_addr)/binary, (?node2_addr)/binary>>,
    ExpectedHeader =
        <<?IPHC_DHTYPE:3, 3:2, 0:1, 2:2, 0:1, 0:1, 1:2, 0:1, 0:1, 1:2, InlineData/
            binary>>,

    % Create the IPHC packet
    {IPHC, _} = lowpan_core:compressIpv6Header(?Ipv6Pckt, false),
    io:format("IPHC: ~p~n", [IPHC]),
    io:format("ExpectedHeader: ~p~n", [ExpectedHeader]),
    IPHC = ExpectedHeader.

msh_pckt(_Config) ->
```

```
    MeshHeader =
        #mesh_header{
            v_bit = 0,
            f_bit = 0,
            hops_left = 14,
            originator_address = ?Node1MacAddress,
            final_destination_address = ?Node2MacAddress
        },

    BinMeshHeader = lowpan_core:buildMeshHeader(MeshHeader),
    ExpectedHeader =
        <<?MESH_DHTYPE:2, 0:1, 0:1, 14:4, ?Node1MacAddress/binary, ?Node2MacAddress/binary>>,

    ExpectedHeader = BinMeshHeader.


broadcast_pckt(_Config) ->
    DestMacAddr = lowpan_core:generateEUI64MacAddr(<<16#1234:16>>),
    MeshHeader =
        #mesh_header{
            v_bit = 0,
            f_bit = 0,
            hops_left = 14,
            originator_address = ?Node1MacAddress,
            final_destination_address = DestMacAddr
        },

    BinMeshHeader = lowpan_core:buildMeshHeader(MeshHeader),

    DestAddr = <<16#FF02:16, 0:64, 1:16, 16#FF00:16, 16#1234:16>>,
    DestAddress = binary:decode_unsigned(DestAddr),
    {_, BroadcastHeader, _} = lowpan_core:getNextHop(?Node1MacAddress, ?Node1MacAddress, DestMacAddr, DestAddress, 3, false),

    ExpectedHeader = <<BinMeshHeader/bitstring, ?BC0_DHTYPE, 3:8>>,

    io:format("Expected: ~p~n~nReceived: ~p~n", [ExpectedHeader, BroadcastHeader]),
    ExpectedHeader = BroadcastHeader.

%-------------------------------------------------------------------------------
%                          Ipv6 Packet Compression
%-------------------------------------------------------------------------------

%--- Basic IPHC test case

% Link-local address
link_local_addr_pckt_comp(_Config) ->
    Payload = <<"Testing basic IPHC compression with link-local address">>,
    IPv6Header =
        #ipv6_header{
            version = 6,
            traffic_class = 0,
            flow_label = 0,
            payload_length = byte_size(Payload),
            next_header = 0,
            hop_limit = 64,
            source_address = <<16#FE80:16, 0:48, ?Node1MacAddress/binary>>,
            destination_address = <<16#FE80:16, 0:48, ?Node2MacAddress/binary>>
        },
    Ipv6Pckt = ipv6:buildIpv6Packet(IPv6Header, Payload),
```

```
    Tf = 2#11,
    Nh = 0,
    Hlim = 2#10,
    Cid = 0,
    Sac = 0,
    Sam = 2#01,
    M = 0,
    Dac = 0,
    Dam = 2#01,
    ExpectedCarriedInline =
        #{
            "SAM" => <<?Node1MacAddress/binary>>,
            "DAM" => <<?Node2MacAddress/binary>>,
            "NextHeader" => 0
        },

    InlineData =
        <<0:8, ?Node1MacAddress/binary,
            ?Node2MacAddress/binary>>,
    ExpectedHeader =
        <<?IPHC_DHTYPE:3, Tf:2, Nh:1, Hlim:2, Cid:1, Sac:1, Sam:2, M:1, Dac:1, Dam
            :2, InlineData/binary>>,

    {CompressedHeader, CarriedInlineData} = lowpan_core:compressIpv6Header(
        Ipv6Pckt, false),
    io:format("Expected ~p~nReceived ~p~n", [ExpectedHeader, CompressedHeader]),
    ExpectedHeader = CompressedHeader,

    ExpectedCarriedInline = CarriedInlineData,
    ok.

% Multicast address
multicast_addr_pckt_comp(_Config) ->
    Payload = <<"Testing basic IPHC compression with multicast address">>,
    IPv6Header =
        #ipv6_header{
            version = 6,
            traffic_class = 0,
            flow_label = 2,
            payload_length = byte_size(Payload),
            %UDP
            next_header = 0,
            hop_limit = 1,
            source_address = <<16#FE80:16, 0:48, ?Node1MacAddress/binary>>,
            destination_address = <<16#FF02:16, 0:48, ?Node2MacAddress/binary>>
        },

    Ipv6Pckt = ipv6:buildIpv6Packet(IPv6Header, Payload),

    Tf = 2#01,
    Nh = 0,
    Hlim = 2#01,
    Cid = 0,
    Sac = 0,
    Sam = 2#01,
    M = 1,
    Dac = 0,
    Dam = 2#00,

    Dest = IPv6Header#ipv6_header.destination_address,
    ExpectedCarriedInline =
```

```
        #{
            "SAM" => <<?Node1MacAddress/binary>>,
            "DAM" => <<Dest/binary>>,
            "NextHeader" => 0,
            "ECN" => 0,
            "FlowLabel" => 2
        },


    InlineData =
        <<0:2, 0:2, 2:20, 0:8, ?Node1MacAddress/binary,
            Dest/binary>>,

    ExpectedHeader =
        <<?IPHC_DHTYPE:3, Tf:2, Nh:1, Hlim:2, Cid:1, Sac:1, Sam:2, M:1, Dac:1, Dam:2, InlineData/binary>>,

    {CompressedHeader, CarriedInlineData} = lowpan_core:compressIpv6Header(
        Ipv6Pckt, false),
    io:format("Expected ~p~nReceived ~p~n", [ExpectedHeader, CompressedHeader]),
    ExpectedHeader = CompressedHeader,

    ExpectedCarriedInline = CarriedInlineData,
    ok.

%---Global contexts test case, affected fields are cid, sac and dac
global_context_pckt_comp1(_Config) ->
    Payload = <<"Testing basic IPHC compression with multicast address">>,
    Source_address = <<16#2001:16, 0:48, ?Node1MacAddress/binary>>,
    Destination_address = <<16#2001:16, 0:48, ?Node2MacAddress/binary>>,
    IPv6Header =
        #ipv6_header{
            version = 6,
            traffic_class = 0,
            flow_label = 3,
            payload_length = byte_size(Payload),
            %UDP
            next_header = 0,
            hop_limit = 255,
            source_address = Source_address,
            destination_address = Destination_address
        },

    Ipv6Pckt = ipv6:buildIpv6Packet(IPv6Header, Payload),

    Tf = 2#01,
    Nh = 0,
    Hlim = 2#11,
    Cid = 0,
    Sac = 1,
    Sam = 2#00,
    M = 0,
    Dac = 1,
    Dam = 2#00,

    ExpectedCarriedInline =
        #{
            "SAM" => Source_address,
            "NextHeader" => 0,
            "ECN" => 0,
            "FlowLabel" => 3,
            "DAM" => Destination_address
```

```
        },
    io:format("ExpectedCarriedInline: ~p~n", [ExpectedCarriedInline]),

    InlineData =
        <<0:2, 0:2, 3:20, 0:8, Source_address/binary, Destination_address/binary
            >>,
    ExpectedHeader =
        <<?IPHC_DHTYPE:3, Tf:2, Nh:1, Hlim:2, Cid:1, Sac:1, Sam:2, M:1, Dac:1, Dam
            :2, InlineData/binary>>,

    {CompressedHeader, CarriedInlineData} = lowpan_core:compressIpv6Header(
        Ipv6Pckt, false),
    io:format("Expected ~p~nReceived ~p~n", [ExpectedHeader, CompressedHeader]),
    ExpectedHeader = CompressedHeader,

    ExpectedCarriedInline = CarriedInlineData,
    ok.

%---Different types of Next Headers test case
udp_nh_pckt_comp(_Config) ->
    Payload = <<"Testing basic IPHC compression with link-local address">>,

    PayloadLength = byte_size(Payload),
    Source_address = <<16#FE80:16, 0:48, ?Node1MacAddress/binary>>,
    Destination_address = <<16#FE80:16, 0:48, ?Node2MacAddress/binary>>,

    UdpPckt = <<1025:16, 61617:16, 25:16, 16#f88c:16>>,

    Ipv6Pckt =
        <<6:4, 0:8, 0:20, PayloadLength:16, 17:8, 64:8, Source_address/binary,
            Destination_address/binary, UdpPckt/binary, Payload/binary>>,

    Tf = 2#11,
    Nh = 1,
    Hlim = 2#10,
    Cid = 0,
    Sac = 0,
    Sam = 2#01,
    M = 0,
    Dac = 0,
    Dam = 2#01,
    C = 0,
    P = 2#01,
    ExpectedCarriedInline = #{"SAM" => <<?Node1MacAddress/binary>>, "DAM" => <<?
        Node2MacAddress/binary>>},

    InlineData = <<?Node1MacAddress/binary, ?Node2MacAddress/binary>>,
    UdpInline = <<1025:16, 177:8, 63628:16>>,

    io:format("UdpInline ~p~n", [UdpInline]),
    ExpectedHeader =
        <<?IPHC_DHTYPE:3, Tf:2, Nh:1, Hlim:2, Cid:1, Sac:1, Sam:2, M:1, Dac:1, Dam
            :2, InlineData/binary, ?UDP_DHTYPE:5, C:1, P:2, UdpInline/binary>>,

    Pckt = <<Ipv6Pckt/binary, UdpPckt/binary>>,
    {CompressedHeader, CarriedInlineData} = lowpan_core:compressIpv6Header(Pckt,
        false),

    io:format("Expected ~p~nReceived ~p~n", [ExpectedHeader, CompressedHeader]),
    ExpectedHeader = CompressedHeader,

    ExpectedCarriedInline = CarriedInlineData,
```

```
    ok.

tcp_nh_pckt_comp(_Config) ->
    Payload = <<"Testing basic IPHC compression with link-local address">>,
    IPv6Header =
        #ipv6_header{
            version = 6,
            traffic_class = 0,
            flow_label = 0,
            payload_length = byte_size(Payload),
            % TCP
            next_header = 6,
            hop_limit = 64,
            source_address = <<16#FE80:16, 0:48, ?Node1MacAddress/binary>>,
            destination_address = <<16#FE80:16, 0:48,?Node2MacAddress/binary>>
        },

    Ipv6Pckt = ipv6:buildIpv6Packet(IPv6Header, Payload),

    Tf = 2#11,
    Nh = 0,
    Hlim = 2#10,
    Cid = 0,
    Sac = 0,
    Sam = 2#01,
    M = 0,
    Dac = 0,
    Dam = 2#01,
    ExpectedCarriedInline =
        #{
            "SAM" => <<?Node1MacAddress/binary>>,
            "DAM" => <<?Node2MacAddress/binary>>,
            "NextHeader" => 6
        },

    InlineData = <<6:8, ?Node1MacAddress/binary, ?Node2MacAddress/binary>>,
    ExpectedHeader =
        <<?IPHC_DHTYPE:3, Tf:2, Nh:1, Hlim:2, Cid:1, Sac:1, Sam:2, M:1, Dac:1, Dam
            :2, InlineData/binary>>,

    {CompressedHeader, CarriedInlineData} = lowpan_core:compressIpv6Header(
        Ipv6Pckt, false),

    io:format("Expected ~p~nReceived ~p~n", [ExpectedHeader, CompressedHeader]),
    ExpectedHeader = CompressedHeader,

    ExpectedCarriedInline = CarriedInlineData,
    ok.

icmp_nh_pckt_comp(_Config) ->
    Payload = <<"Testing basic IPHC compression with link-local address">>,
    IPv6Header =
        #ipv6_header{
            version = 6,
            traffic_class = 0,
            flow_label = 0,
            payload_length = byte_size(Payload),
            %ICMPv6
            next_header = 58,
            hop_limit = 255,
            source_address = <<16#FE80:16, 0:48, ?Node1MacAddress/binary>>,
            destination_address = <<16#FE80:16, 0:48, ?Node2MacAddress/binary>>
```

```
        },

    Ipv6Pckt = ipv6:buildIpv6Packet(IPv6Header, Payload),

    Tf = 2#11,
    Nh = 0,
    Hlim = 2#11,
    Cid = 0,
    Sac = 0,
    Sam = 2#01,
    M = 0,
    Dac = 0,
    Dam = 2#01,
    ExpectedCarriedInline =
        #{
            "SAM" => <<?Node1MacAddress/binary>>,
            "DAM" => <<?Node2MacAddress/binary>>,
            "NextHeader" => 58
        },

    InlineData = <<58:8, ?Node1MacAddress/binary, ?Node2MacAddress/binary>>,
    ExpectedHeader =
        <<?IPHC_DHTYPE:3, Tf:2, Nh:1, Hlim:2, Cid:1, Sac:1, Sam:2, M:1, Dac:1, Dam
            :2, InlineData/binary>>,

    {CompressedHeader, CarriedInlineData} = lowpan_core:compressIpv6Header(
        Ipv6Pckt, false),

    io:format("Expected ~p~nReceived ~p~n", [ExpectedHeader, CompressedHeader]),
    ExpectedHeader = CompressedHeader,

    ExpectedCarriedInline = CarriedInlineData,
    ok.

%---Online resource (https://www.youtube.com/watch?v=0JMVO3HN0xo&t=778s)
compress_header_example1_test(_Config) ->
    Payload = <<"Hello world this is an ipv6 packet">>,
    PayloadLength = byte_size(Payload),

    SrcAddress = <<16#FE80:16, 0:48, 16#020164FFFE2FFC0A:64>>,
    DstAddress = <<16#FF02:16, 0:48, 16#0000000000000001:64>>,
    Ipv6Pckt =
        <<6:4, 224:8, 0:20, PayloadLength:16, 58:8, 255:8, SrcAddress/binary,
            DstAddress/binary, Payload/bitstring>>,

    Tf = 2#10,
    Nh = 0,
    Hlim = 2#11,
    Cid = 0,
    Sac = 0,
    Sam = 2#11,
    M = 1,
    Dac = 0,
    Dam = 2#11,
    ExpectedCarriedInline =
        #{
            "DAM" => <<1>>,
            "NextHeader" => 58,
            "TrafficClass" => 224
        },
    InlineData = <<0:2, 56:6, 58:8, 1:8>>,
    ExpectedHeader =
```

```
        <<?IPHC_DHTYPE:3, Tf:2, Nh:1, Hlim:2, Cid:1, Sac:1, Sam:2, M:1, Dac:1, Dam:2, InlineData/binary>>,

    {CompressedHeader, CarriedInlineData} = lowpan_core:compressIpv6Header(Ipv6Pckt, false),
    io:format("Expected ~p~nReceived ~p~n", [ExpectedHeader, CompressedHeader]),

    ExpectedHeader = CompressedHeader,

    ExpectedCarriedInline = CarriedInlineData,
    ok.

compress_header_example2_test(_Config) ->
    Payload = <<"Hello world this is an ipv6 packet">>,
    PayloadLength = byte_size(Payload),

    SrcAddress = <<16#2001066073013728:64, 16#0223DFFFFEA9F7AC:64>>,
    DstAddress = <<16#2001A45040070803:64, 16#0000000000001004:64>>,
    Ipv6Pckt =
        <<6:4, 0:8, 0:20, PayloadLength:16, 6:8, 64:8, SrcAddress/binary, DstAddress/binary, Payload/binary>>,

    Tf = 2#11,
    Nh = 0,
    Hlim = 2#10,
    Cid = 0,
    Sac = 1,
    Sam = 2#00,
    M = 0,
    Dac = 1,
    Dam = 2#00,
    InlineData = <<6:8, SrcAddress/binary, DstAddress/binary>>,
    ExpectedHeader =
        <<?IPHC_DHTYPE:3, Tf:2, Nh:1, Hlim:2, Cid:1, Sac:1, Sam:2, M:1, Dac:1, Dam:2, InlineData/binary>>,

    {CompressedHeader, _} = lowpan_core:compressIpv6Header(Ipv6Pckt, false),
    io:format("Expected ~p~nActual ~p~n", [ExpectedHeader, CompressedHeader]),

    ExpectedHeader = CompressedHeader,

    ok.

%-------------------------------------------------------------------------------
%                         6LoWPAN IPv6 Packet Fragmentation
%-------------------------------------------------------------------------------

fragmentation_test(_Config) ->
    % fragmentation test based on the computation of the size of all fragment payloads
    Payload = <<"This is an Ipv6 pckt">>,
    IPv6Header =
        #ipv6_header{
            version = 6,
            traffic_class = 0,
            flow_label = 0,
            payload_length = byte_size(Payload),
            next_header = 17,
            hop_limit = 64,
            source_address = <<1>>,
            destination_address = <<2>>
        },
```

```
    IPv6Pckt = ipv6:buildIpv6Packet(IPv6Header, Payload),
    Fragments = lowpan_core:fragmentIpv6Packet(IPv6Pckt, byte_size(Payload), false
        ),
    ReassembledSize =
        lists:foldl(fun({_, Fragment}, Acc) -> byte_size(Fragment) + Acc end, 0,
            Fragments),
    Psize = byte_size(IPv6Pckt),
    Psize = ReassembledSize,
    ok.

datagram_info_test(_Config) ->
    Data = <<"payload">>,
    Fragment = <<?FRAG1_DHTYPE:5, 1000:11, 12345:16, Data/bitstring>>,

    DtgInfo = lowpan_core:datagramInfo(Fragment),
    FragType = DtgInfo#datagramInfo.fragtype,
    DatagramSize = DtgInfo#datagramInfo.datagramSize,
    DatagramTag = DtgInfo#datagramInfo.datagramTag,
    DatagramOffset = DtgInfo#datagramInfo.datagramOffset,
    Payload = DtgInfo#datagramInfo.payload,

    io:format("~p~n", [Payload]),

    ?FRAG1_DHTYPE = FragType,
    1000 = DatagramSize,
    12345 = DatagramTag,
    0 = DatagramOffset,
    Data = Payload,
    ok.

%--------------------------------------------------------------------------------
%                           Ipv6 Packet Reassembly
%--------------------------------------------------------------------------------

reassemble_fragments_list_test(_Config) ->
    Data = <<"Hello World!">>,
    PayloadLen = byte_size(Data),
    Datagram = #datagram{
        tag = 25,
        size = PayloadLen,
        cmpt = PayloadLen,
        fragments = #{0 => <<"Hello ">>, 1 => <<"World!">>},
        timer = erlang:system_time(second)
    },

    Reassembled = lowpan_core:reassemble(Datagram),
    <<"Hello World!">> = Reassembled,
    ok.

reassemble_single_fragments_test(_Config) ->
    Data = <<"Hello World!">>,
    PayloadLen = byte_size(Data),

    DatagramMap = ets:new(datagram_map_test, [named_table, public]),
    {Result1, _Map1} = lowpan_core:storeFragment(DatagramMap, {<<1>>, 25}, 0, <<"
        Hello ">>, erlang:system_time(second), PayloadLen, 25, self()),
    incomplete_first = Result1,

    {Result2, _Map2} = lowpan_core:storeFragment(DatagramMap, {<<1>>, 25}, 1, <<"
        World!">>, erlang:system_time(second), PayloadLen, 25, self()),
    complete = Result2,

```

```
    Reassembled = lowpan_core:reassemble(#datagram{
                                    tag = 25,
                                    size = PayloadLen,
                                    cmpt = PayloadLen,
                                    timer = erlang:system_time(second),
                                    fragments = #{0 => <<"Hello ">>, 1 => <<"
                                        World!">>}
                                }),
    Data = Reassembled,
    ets:delete(DatagramMap),
    ok.

reassemble_full_ipv6_pckt_test(_Config) ->
    Payload = lowpan_core:generateChunks(),
    IPv6Header =
        #ipv6_header{
            version = 6,
            traffic_class = 0,
            flow_label = 0,
            payload_length = byte_size(Payload),
            next_header = 17,
            hop_limit = 64,
            source_address = <<1:128>>,
            destination_address = <<2:128>>
        },

    Ipv6Pckt = ipv6:buildIpv6Packet(IPv6Header, Payload),
    io:format("Original pckt size ~p bytes~n", [byte_size(Ipv6Pckt)]),
    FragmentList = lowpan_core:fragmentIpv6Packet(Ipv6Pckt, byte_size(Ipv6Pckt),
        false),

    DatagramMap = ets:new(datagram_map_test, [named_table, public]),

    lists:foreach(
        fun({FragHeader, FragPayload}) ->
            Offset = case byte_size(FragHeader) of
                4 -> %first frag
                    0;
                5 ->
                    <<_:32, Ofset:8>> = FragHeader,
                    Ofset
            end,

            io:format("Storing fragment with offset ~p~n", [Offset]),
            {Result, _Map} = lowpan_core:storeFragment(DatagramMap, {<<1>>, 25},
                Offset, FragPayload, erlang:system_time(second), byte_size(
                Ipv6Pckt), 25, self()),
            io:format("Fragment stored result: ~p~n", [Result])
        end,
        FragmentList
    ),

    Datagram = ets:lookup_element(DatagramMap, {<<1>>, 25}, 2),
    io:format("Datagram after storing fragments: ~p~n", [Datagram]),

    Reassembled = lowpan_core:reassemble(Datagram),
    io:format("Reassembled: ~p~nIpv6Pckt: ~p~n", [Reassembled, Ipv6Pckt]),

    case Ipv6Pckt of
        Reassembled -> io:format("Reassembly successful.~n");
        _ -> io:format("Reassembly failed.~n")
    end,
```

```erlang

    % Nettoyage
    ets:delete(DatagramMap),
    ok.


%-----------------------------------------------------------------------------------
%                                 Additionnal tests
%-----------------------------------------------------------------------------------
extended_EUI64_from_48mac(_Config)->
    MacAddr = <<16#9865FD361453:48>>,
    Expected = <<16#9A65FDFFFE361453:64>>,
    Result = lowpan_core:getEUI64From48bitMac(MacAddr),
    io:format("Expected ~p~nResult ~p~n",[Expected, Result]),
    Result = Expected.

extended_EUI64_from_64mac(_Config)->
    MacAddr = <<16#00124B0006386C1A:64>>,
    Expected = <<16#02124B0006386C1A:64>>,
    Result = lowpan_core:getEUI64FromExtendedMac(MacAddr),
    io:format("Expected ~p~nResult ~p~n",[Expected, Result]),
    Result = Expected.

extended_EUI64_from_16mac(_Config)->
    MacAddr = <<16#0001:16>>,
    Expected = <<16#FDFF:16, 0:8, 16#FFFE:16, 0:8, 16#0001:16>>,
    Result = lowpan_core:getEUI64FromShortMac(MacAddr),
    io:format("Expected ~p~nResult ~p~n",[Expected, Result]),
    Result = Expected.

link_local_from_16mac(_Config)->
    MacAddr = <<16#0001:16>>,
    Expected = <<16#FE80:16, 0:48,16#FDFF:16, 0:8, 16#FFFE:16, 0:8, 16#0001:16>>,
    Result = lowpan_core:generateLLAddr(MacAddr),
    io:format("Expected ~p~nResult ~p~n",[Expected, Result]),
    Result = Expected.

check_tag_unicity(_Config) ->
    TagMap = #{},
    Tag1 = 10,
    {_, NewTagMap} = lowpan_core:checkTagUnicity(TagMap, Tag1),
    Tag2 = 10,
    {ValidTag2, FinalMap} = lowpan_core:checkTagUnicity(NewTagMap, Tag2),
    io:format("TagMap: ~p~n", [FinalMap]),
    ValidTag2 =/= Tag2.


multicast_addr_validity(_Config) ->
    Ipv6Addr = <<16#FF02:16, 0:64, 1:16, 16#FF00:16, 16#1234:16>>,
    GenAddr = lowpan_core:generateMulticastAddr(Ipv6Addr),
    ExpectedAddr = <<16#9234:16>>,
    io:format("ExpectedAddr ~p~nGenAddr ~p~n", [ExpectedAddr, GenAddr]),
    GenAddr = ExpectedAddr.



%
    ----------------------------------------------------------------------------------------------------------------------
%                                                 Contiki-ng cooja packet
    example tests
%
    ----------------------------------------------------------------------------------------------------------------------
```

```
cooja_example1(_Config)->
    Payload = <<"Cooja example 1">>,
    PayloadLength = byte_size(Payload),

    Source_address = <<16#FE80:16, 0:48, 207:16, 7:16, 7:16, 7:16>>,
    Destination_address = <<16#FF02:16, 0:48, 0:48,16#1a:16>>,

    Ipv6Pckt =
        <<6:4, 0:8, 0:20, PayloadLength:16, 58:8, 64:8, Source_address/binary,
            Destination_address/binary, Payload/binary>>,

    Tf = 2#11,
    Nh = 0,
    Hlim = 2#10,
    Cid = 0,
    Sac = 0,
    Sam = 2#11,
    M = 1,
    Dac = 0,
    Dam = 2#11,
    <<_:120,Last8:8>> = Destination_address,
    ExpectedCarriedInline = #{"NextHeader" => 58, "DAM"=><<Last8:8>>},

    InlineData = <<58:8, Last8:8>>,
    UdpInline = <<1025:16, 177:8, 63628:16>>,

    %io:format("Expected UdpInline ~p~n", [UdpInline]),
    ExpectedHeader =
        <<?IPHC_DHTYPE:3, Tf:2, Nh:1, Hlim:2, Cid:1, Sac:1, Sam:2, M:1, Dac:1, Dam
            :2, InlineData/binary>>,

    CH = {Tf, Nh, Hlim, Cid, Sac, Sam, M, Dac, Dam, InlineData},
    io:format("CH: ~p~n",[CH]),
    io:format("Expected carried values: ~p~n", [ExpectedCarriedInline]),
    Pckt = <<Ipv6Pckt/binary>>,
    {CompressedHeader, CarriedInlineData} = lowpan_core:compressIpv6Header(Pckt,
        false),



    %io:format("Expected ~p~nReceived ~p~n", [ExpectedHeader, CompressedHeader]),
    ExpectedHeader = CompressedHeader,

    ExpectedCarriedInline = CarriedInlineData,
    ok.

cooja_example2(_Config)->
    Payload = <<"Cooja example 2">>,
    PayloadLength = byte_size(Payload),

    Source_address = <<16#FE80:16, 0:48, 202:16, 2:16, 2:16, 2:16>>,
    Destination_address = <<16#FE80:16, 0:48, 212:16, 7402:16, 2:16, 2:16>>,

    UdpPckt = <<5683:16, 5683:16, 37:16, 16#8441:16>>,

    Ipv6Pckt =
        <<6:4, 0:8, 0:20, PayloadLength:16, 17:8, 64:8, Source_address/binary,
            Destination_address/binary, UdpPckt/binary, Payload/binary>>,

    Tf = 2#11,
    Nh = 1,
```

```
    Hlim = 2#10,
    Cid = 0,
    Sac = 0,
    Sam = 2#11,
    M = 0,
    Dac = 0,
    Dam = 2#11,
    C = 0,
    P = 2#00,
    ExpectedCarriedInline = #{},

    %InlineData = <<?Node1MacAddress/binary, ?Node2MacAddress/binary>>,
    UdpInline = <<5683:16, 5683:8, 8441:16>>,

    %io:format("Expected UdpInline ~p~n", [UdpInline]),
    ExpectedHeader =
        <<?IPHC_DHTYPE:3, Tf:2, Nh:1, Hlim:2, Cid:1, Sac:1, Sam:2, M:1, Dac:1, Dam
            :2, ?UDP_DHTYPE:5, C:1, P:2, UdpInline/binary>>,

    CH = {Tf, Nh, Hlim, Cid, Sac, Sam, M, Dac, Dam, UdpInline},
    io:format("Expected carried values: ~p~n", [ExpectedCarriedInline]),
    Pckt = <<Ipv6Pckt/binary, UdpPckt/binary>>,
    {CompressedHeader, CarriedInlineData} = lowpan_core:compressIpv6Header(Pckt,
        false),

    %io:format("Expected ~p~nReceived ~p~n", [ExpectedHeader, CompressedHeader]),
    ExpectedHeader = CompressedHeader,

    ExpectedCarriedInline = CarriedInlineData,
    ok.

cooja_example3(_Config)->
    Payload = <<"Cooja example 3">>,
    PayloadLength = byte_size(Payload),

    Source_address = <<0:64, 207:16, 7:16, 7:16, 7:16>>,
    Destination_address = <<0:64, 203:16, 3:16, 3:16, 3:16>>,


    Ipv6Pckt =
        <<6:4, 0:8, 0:20, PayloadLength:16, 43:8, 63:8, Source_address/binary,
            Destination_address/binary, Payload/binary>>,

    Tf = 2#11,
    Nh = 1,
    Hlim = 2#00,
    Cid = 1,
    Sac = 1,
    Sam = 2#01,
    M = 0,
    Dac = 1,
    Dam = 2#11,

    <<_:64, Last64S:64>> = Source_address,

    ExpectedCarriedInline = #{"HopLimit"=>63, "NH"=>43, "SAM" => <<Last64S:64>>},

    InlineData = <<?Node1MacAddress/binary, ?Node2MacAddress/binary>>,
    UdpInline = <<1025:16, 177:8, 63628:16>>,

    %io:format("Expected UdpInline ~p~n", [UdpInline]),
    ExpectedHeader =
```

```
        <<?IPHC_DHTYPE:3, Tf:2, Nh:1, Hlim:2, Cid:1, Sac:1, Sam:2, M:1, Dac:1, Dam
            :2, InlineData/binary>>,

    CH = {Tf, Nh, Hlim, Cid, Sac, Sam, M, Dac, Dam, InlineData},
    io:format("Expected carried values: ~p~n", [ExpectedCarriedInline]),
    Pckt = <<Ipv6Pckt/binary>>,

    {CompressedHeader, CarriedInlineData} = lowpan_core:compressIpv6Header(Pckt,
        false),

    %io:format("Expected ~p~nReceived ~p~n", [ExpectedHeader, CompressedHeader]),
    ExpectedHeader = CompressedHeader,

    ExpectedCarriedInline = CarriedInlineData,
    ok.
```

A.10 Simulation tests code

```
-module(lowpan_sender_receiver_SUITE).

-include_lib("common_test/include/ct.hrl").
-include("../src/utils.hrl").

-export([
    all/0, groups/0, init_per_suite/1, end_per_suite/1, init_per_group/2,
    end_per_group/2, init_per_testcase/2, end_per_testcase/2,
    simple_pckt_sender/1, simple_pckt_receiver/1, big_payload_sender/1,
    big_payload_receiver/1, multicast_sender/1, unspecified_dst_sender/1,
        routing_req_sender/1, routing_req_receiver2/1,
    routing_req_receiver3/1, big_pyld_routing_sender/1, big_pyld_routing_receiver2
        /1,
    big_pyld_routing_receiver3/1, discarded_sender/1, discarded_receiver/1,
    no_hoplft_dst_reached_sender/1, no_hoplft_dst_reached_receiver/1,
    unexpected_dtg_size_sender/1,same_tag_different_senders_sender/1,
    same_tag_different_senders_receiver/1,
    timeout_sender/1, timeout_receiver/1, duplicate_sender/1,
    duplicate_receiver/1, multiple_hop_sender/1, multiple_hop_receiver2/1,
        multiple_hop_receiver3/1, multiple_hop_receiver4/1,
    nalp_sender/1, broadcast_sender/1, broadcast_receiver/1,
        extended_hopsleft_sender/1, extended_hopsleft_receiver2/1,
    extended_hopsleft_receiver3/1, extended_hopsleft_receiver4/1,
        mesh_prefix_sender/1, mesh_prefix_receiver/1,
    simple_udp_pckt_sender/1, simple_udp_pckt_receiver/1
]).

all() ->
    [{group, test_scenarios}].

%---------- Tests groups
    -------------------------------------------------------------

groups() ->
    [
        {test_scenarios, [], [
            {group, simple_tx_rx},
            {group, big_payload_tx_rx},
            {group, multicast_src_tx},
```

```
            {group, unspecified_dst_tx},
            {group, routing_req_tx_rx},
            {group, discard_datagram_tx_rx},
            {group, no_hoplft_dst_reached_tx_rx},
            {group, unexpected_dtg_size_tx},
            {group, same_tag_different_senders},
            {group, timeout_scenario},
            {group, duplicate_tx_rx},
            {group, multiple_hop_tx_rx},
            {group, nalp_tx_rx},
            {group, broadcast_tx_rx},
            {group, extendedHopsleftTx_rx},
            {group, big_pyld_routing_tx_rx},
            {group, simple_udp_tx_rx},
            {group, mesh_prefix_tx_rx}
        ]},
        {simple_tx_rx, [parallel, {repeat, 1}], [simple_pckt_sender,
            simple_pckt_receiver]},
        {simple_udp_tx_rx, [parallel, {repeat, 1}], [simple_udp_pckt_sender,
            simple_udp_pckt_receiver]},
        {big_payload_tx_rx, [parallel, {repeat, 1}], [big_payload_sender,
            big_payload_receiver]},
        {multicast_src_tx, [sequential], [multicast_sender]},
        {unspecified_dst_tx, [sequential], [unspecified_dst_sender]},
        {routing_req_tx_rx, [parallel, {repeat, 1}], [routing_req_sender,
            routing_req_receiver3, routing_req_receiver2]},
        {big_pyld_routing_tx_rx, [parallel, {repeat, 1}], [big_pyld_routing_sender
            , big_pyld_routing_receiver2, big_pyld_routing_receiver3]},
        {discard_datagram_tx_rx, [parallel, {repeat, 1}], [discarded_sender,
            discarded_receiver]},
        {no_hoplft_dst_reached_tx_rx, [parallel, {repeat, 1}], [
            no_hoplft_dst_reached_sender, no_hoplft_dst_reached_receiver]},
        {unexpected_dtg_size_tx, [sequential], [unexpected_dtg_size_sender]},
        {same_tag_different_senders, [parallel, {repeat, 1}], [
            same_tag_different_senders_sender, same_tag_different_senders_receiver
            ]},
        {timeout_scenario, [parallel, {repeat, 1}], [timeout_sender,
            timeout_receiver]},
        {duplicate_tx_rx, [parallel, {repeat, 1}], [duplicate_sender,
            duplicate_receiver]},
        {multiple_hop_tx_rx, [parallel, {repeat, 1}], [multiple_hop_sender,
            multiple_hop_receiver2, multiple_hop_receiver3, multiple_hop_receiver4
            ]},
        {nalp_tx_rx, [sequential], [nalp_sender]},
        {broadcast_tx_rx, [parallel, {repeat, 1}], [broadcast_sender,
            broadcast_receiver]},
        {extendedHopsleftTx_rx, [parallel, {repeat, 1}], [extended_hopsleft_sender
            , extended_hopsleft_receiver2, extended_hopsleft_receiver3,
            extended_hopsleft_receiver4]},
        {mesh_prefix_tx_rx, [parallel, {repeat, 1}], [mesh_prefix_sender,
            mesh_prefix_receiver]}
    ].

%--------------------------
init_per_group(simple_tx_rx, Config) ->
    init_per_group_setup(?Node1Address, ?Node2Address, ?Payload, Config);
%--------------------------
init_per_group(simple_udp_tx_rx, Config) ->
    init_per_group_udp_setup(?Node1Address, ?Node2Address, <<"Hello world">>,
        Config);
%--------------------------
init_per_group(big_payload_tx_rx, Config) ->
```

```
    init_per_group_setup(?Node1Address, ?Node2Address, ?BigPayload, Config);
%--------------------------
init_per_group(multicast_src_tx, Config) ->
    init_per_group_setup(<<16#FF:16, 0:112>>, ?Node2Address, ?Payload, Config);
%--------------------------
init_per_group(unspecified_dst_tx, Config) ->
    init_per_group_setup(?Node1Address, <<0:128>>, ?Payload, Config);
%--------------------------
init_per_group(routing_req_tx_rx, Config) ->
    init_per_group_setup(?Node1Address, ?Node2Address, ?Payload, Config);
%--------------------------
init_per_group(big_pyld_routing_tx_rx, Config) ->
    init_per_group_setup(?Node1Address, ?Node3Address, ?BigPayload, Config);
%--------------------------
init_per_group(discard_datagram_tx_rx, Config) ->
    init_per_group_setup(?Node1Address, ?Node2Address, ?Payload, Config);
    %--------------------------
init_per_group(no_hoplft_dst_reached_tx_rx, Config) ->
    init_per_group_setup(?Node1Address, ?Node2Address, ?Payload, Config);
%--------------------------
init_per_group(unexpected_dtg_size_tx, Config) ->
    Payload = lowpan_core:generateChunks(120),
    init_per_group_setup(?Node1Address, ?Node2Address, Payload, Config);
%--------------------------
init_per_group(same_tag_different_senders, Config) ->
    init_per_group_setup(?Node1Address, ?Node2Address, ?Payload, Config);
%--------------------------
init_per_group(timeout_scenario, Config) ->
    init_per_group_setup(?Node1Address, ?Node2Address, ?Payload, Config);
%--------------------------
init_per_group(tag_verification_tx_rx, Config) ->
    init_per_group_setup(?Node1Address, ?Node2Address, ?BigPayload, Config);
%--------------------------
init_per_group(duplicate_tx_rx, Config) ->
    init_per_group_setup(?Node1Address, ?Node2Address, ?Payload, Config);
%--------------------------
init_per_group(multiple_hop_tx_rx, Config) ->
    init_per_group_setup(?Node1Address, ?Node4Address, ?Payload, Config);
%--------------------------
init_per_group(nalp_tx_rx, Config) ->
    init_per_group_setup(?Node1Address, ?Node2Address, ?Payload, Config);
%--------------------------
init_per_group(broadcast_tx_rx, Config) ->
    init_per_group_setup(?Node1Address, <<16#FF02:16, 0:64, 1:16, 16#FF00:16,
        16#1234:16>>, ?Payload, Config);
%--------------------------
init_per_group(extendedHopsleftTx_rx, Config) ->
    init_per_group_setup(?Node1Address, ?Node4Address, ?Payload, Config);
%--------------------------
init_per_group(mesh_prefix_tx_rx, Config) ->
    MacAddress = lowpan_core:generateEUI64MacAddr(?Node2MacAddress),
    init_per_group_setup(?Node1Address, <<?MESH_LOCAL_PREFIX:16, 16#0DB8:16, 0:32,
         MacAddress/binary>>, ?Payload, Config);
%--------------------------
init_per_group(benchmark_tx_rx, Config) ->
    init_per_group_setup(?Node1Address, ?Node2Address, ?Payload, Config);
%--------------------------
init_per_group(_, Config) ->
    Config.

init_per_group_setup(Src, Dst, Payload, Config) ->
    {NetPid, Network} = lowpan_node:boot_network_node(#{loss => true}),
```

```
    io:format("Initializing group ~n"),

    IPv6Header = #ipv6_header{
            version = 6,
            traffic_class = 0,
            flow_label = 0,
            payload_length = byte_size(Payload),
            next_header = 12,
            hop_limit = 64,
            source_address = Src,
            destination_address = Dst
        },
    Packet = ipv6:buildIpv6Packet(IPv6Header, Payload),
    [
        {net_pid, NetPid},
        {network, Network},
        {node1_mac_address, ?Node1MacAddress},
        {node2_mac_address, ?Node2MacAddress},
        {node3_mac_address, ?Node3MacAddress},
        {node4_mac_address, ?Node4MacAddress},
        {ipv6_packet, Packet}
        | Config
    ].

init_per_group_udp_setup(Src, Dst, Payload, Config) ->
    {NetPid, Network} = lowpan_node:boot_network_node(#{loss => true}),
    io:format("Initializing group ~n"),

    PayloadLength = byte_size(Payload),
    IPv6Header =
        #ipv6_header{
            version = 6,
            traffic_class = 0,
            flow_label = 0,
            % 4 bytes for the UDP header
            payload_length = PayloadLength,
            next_header = 17,
            hop_limit = 64,
            source_address = Src,
            destination_address = Dst
        },
    UdpHeader =
        #udp_header{
            source_port = 1025,
            destination_port = 61617,
            length = PayloadLength,
            checksum = 16#f88c
        },

    Packet = ipv6:buildIpv6UdpPacket(IPv6Header, UdpHeader, Payload),
    [
        {net_pid, NetPid},
        {network, Network},
        {node1_mac_address, ?Node1MacAddress},
        {node2_mac_address, ?Node2MacAddress},
        {node3_mac_address, ?Node3MacAddress},
        {node4_mac_address, ?Node4MacAddress},
        {ipv6_packet, Packet}
        | Config
    ].

end_per_group(_Group, Config) ->
```

```
    Network = proplists:get_value(network, Config),
    NetPid = proplists:get_value(net_pid, Config),

    if
        Network =:= undefined ->
            io:format("Error: Network not found in Config~n"),
            {error, network_not_found};
        NetPid =:= undefined ->
            io:format("Error: NetPid not found in Config~n"),
            {error, net_pid_not_found};
        true ->
            lowpan_node:stop_network_node(Network, NetPid),
            ok
    end.



%---------- Tests cases initialization
    ------------------------------------------------

defaut_sender_init_per_testcase(Config, RoutingTable)->
    Network = ?config(network, Config),
    Node1MacAddress = ?config(node1_mac_address, Config),
    Node = lowpan_node:boot_lowpan_node(node1, Network, Node1MacAddress,
        RoutingTable),
    [{node1, Node} | Config].

defaut_receiver2_init_per_testcase(Config, RoutingTable)->
    Network = ?config(network, Config),
    Node2MacAddress = ?config(node2_mac_address, Config),
    Callback = fun lowpan_api:inputCallback/4,
    Node = lowpan_node:boot_lowpan_node(node2, Network, Node2MacAddress, Callback,
         RoutingTable),
    [{node2, Node} | Config].

defaut_receiver3_init_per_testcase(Config, RoutingTable)->
    Network = ?config(network, Config),
    Node3MacAddress = ?config(node3_mac_address, Config),
    Callback = fun lowpan_api:inputCallback/4,
    Node = lowpan_node:boot_lowpan_node(node3, Network, Node3MacAddress, Callback,
         RoutingTable),
    [{node3, Node} | Config].

defaut_receiver4_init_per_testcase(Config, RoutingTable)->
    Network = ?config(network, Config),
    Node4MacAddress = ?config(node4_mac_address, Config),
    Callback = fun lowpan_api:inputCallback/4,
    Node = lowpan_node:boot_lowpan_node(node4, Network, Node4MacAddress, Callback,
         RoutingTable),
    [{node4, Node} | Config].


broadcast_receiver_init_per_testcase(Config, RoutingTable)->
    Network = ?config(network, Config),
    MacAddress = <<16#9234:16>>,
    Callback = fun lowpan_api:inputCallback/4,
    Node = lowpan_node:boot_lowpan_node(broadcast_node, Network, MacAddress,
        Callback, RoutingTable),
    [{broadcast_node, Node} | Config].

%--------------------------
init_per_testcase(simple_pckt_sender, Config)->
```

```
    defaut_sender_init_per_testcase(Config, ?Default_routing_table);

init_per_testcase(simple_pckt_receiver, Config)->
    defaut_receiver2_init_per_testcase(Config, ?Default_routing_table);

%--------------------------
init_per_testcase(simple_udp_pckt_sender, Config)->
    defaut_sender_init_per_testcase(Config, ?Default_routing_table);

init_per_testcase(simple_udp_pckt_receiver, Config)->
    defaut_receiver2_init_per_testcase(Config, ?Default_routing_table);

%--------------------------
init_per_testcase(big_payload_sender, Config)->
    defaut_sender_init_per_testcase(Config, ?Default_routing_table);

init_per_testcase(big_payload_receiver, Config)->
    defaut_receiver2_init_per_testcase(Config, ?Default_routing_table);

%--------------------------
init_per_testcase(multicast_sender, Config)->
    Config1 = defaut_sender_init_per_testcase(Config, ?Default_routing_table),
    defaut_receiver2_init_per_testcase(Config1, ?Default_routing_table);

%--------------------------
init_per_testcase(unspecified_dst_sender, Config)->
    Config1 = defaut_sender_init_per_testcase(Config, ?Default_routing_table),
    defaut_receiver2_init_per_testcase(Config1, ?Default_routing_table);

%--------------------------
init_per_testcase(routing_req_sender, Config)->
    defaut_sender_init_per_testcase(Config, ?Node1_routing_table);

init_per_testcase(routing_req_receiver2, Config)->
    defaut_receiver2_init_per_testcase(Config, ?Node2_routing_table);

init_per_testcase(routing_req_receiver3, Config)->
    defaut_receiver3_init_per_testcase(Config, ?Node3_routing_table);

%--------------------------
init_per_testcase(big_pyld_routing_sender, Config)->
    defaut_sender_init_per_testcase(Config, ?Node1_routing_table);

init_per_testcase(big_pyld_routing_receiver2, Config)->
    defaut_receiver2_init_per_testcase(Config, ?Node2_routing_table);

init_per_testcase(big_pyld_routing_receiver3, Config)->
    defaut_receiver3_init_per_testcase(Config, ?Node3_routing_table);

%--------------------------
init_per_testcase(discarded_sender, Config)->
    defaut_sender_init_per_testcase(Config, ?Node1_routing_table);

init_per_testcase(discarded_receiver, Config)->
    defaut_receiver2_init_per_testcase(Config, ?Node2_routing_table);

%--------------------------
init_per_testcase(no_hoplft_dst_reached_sender, Config)->
    defaut_sender_init_per_testcase(Config, ?Node1_routing_table);

init_per_testcase(no_hoplft_dst_reached_receiver, Config)->
    defaut_receiver2_init_per_testcase(Config, ?Node2_routing_table);
```

```

%-------------------------
init_per_testcase(unexpected_dtg_size_sender, Config)->
    defaut_sender_init_per_testcase(Config, ?Node1_routing_table);

%-------------------------
init_per_testcase(same_tag_different_senders_sender, Config) ->
    Config1 = defaut_sender_init_per_testcase(Config, ?Default_routing_table),
    defaut_receiver2_init_per_testcase(Config1, ?Default_routing_table);

init_per_testcase(same_tag_different_senders_receiver, Config) ->
    defaut_receiver3_init_per_testcase(Config, ?Default_routing_table);

%-------------------------
init_per_testcase(timeout_sender, Config) ->
    defaut_sender_init_per_testcase(Config, ?Default_routing_table);
init_per_testcase(timeout_receiver, Config) ->
    defaut_receiver2_init_per_testcase(Config, ?Default_routing_table);

%-------------------------
init_per_testcase(tag_verification_sender, Config) ->
    defaut_sender_init_per_testcase(Config, ?Default_routing_table);

init_per_testcase(tag_verification_receiver, Config) ->
    defaut_receiver2_init_per_testcase(Config, ?Default_routing_table);

%-------------------------
init_per_testcase(duplicate_sender, Config) ->
    defaut_sender_init_per_testcase(Config, ?Default_routing_table);

init_per_testcase(duplicate_receiver, Config) ->
    defaut_receiver2_init_per_testcase(Config, ?Default_routing_table);

%-------------------------
init_per_testcase(multiple_hop_sender, Config) ->
    defaut_sender_init_per_testcase(Config, ?Node1_multiple_hop_routing_table);

init_per_testcase(multiple_hop_receiver2, Config)->
    defaut_receiver2_init_per_testcase(Config, ?Node2_multiple_hop_routing_table);

init_per_testcase(multiple_hop_receiver3, Config)->
    defaut_receiver3_init_per_testcase(Config, ?Node3_multiple_hop_routing_table);

init_per_testcase(multiple_hop_receiver4, Config) ->
    defaut_receiver4_init_per_testcase(Config, ?Node4_multiple_hop_routing_table);

%-------------------------
init_per_testcase(nalp_sender, Config) ->
    defaut_sender_init_per_testcase(Config, ?Default_routing_table);

%-------------------------
init_per_testcase(broadcast_sender, Config) ->
    defaut_sender_init_per_testcase(Config, ?Default_routing_table);

init_per_testcase(broadcast_receiver, Config) ->
    broadcast_receiver_init_per_testcase(Config, ?Default_routing_table);
%-------------------------
init_per_testcase(extended_hopsleft_sender, Config) ->
    defaut_sender_init_per_testcase(Config, ?Node1_multiple_hop_routing_table);

init_per_testcase(extended_hopsleft_receiver2, Config)->
    defaut_receiver2_init_per_testcase(Config, ?Node2_multiple_hop_routing_table);
```

```
init_per_testcase(extended_hopsleft_receiver3, Config)->
    defaut_receiver3_init_per_testcase(Config, ?Node3_multiple_hop_routing_table);

init_per_testcase(extended_hopsleft_receiver4, Config) ->
    defaut_receiver4_init_per_testcase(Config, ?Node4_multiple_hop_routing_table);

%--------------------------
init_per_testcase(mesh_prefix_sender, Config) ->
    defaut_sender_init_per_testcase(Config, ?Default_routing_table);

init_per_testcase(mesh_prefix_receiver, Config) ->
    Network = ?config(network, Config),
    Callback = fun lowpan_api:inputCallback/4,
    Node = lowpan_node:boot_lowpan_node(node2, Network, ?Node2MacAddress, Callback, ?Default_routing_table),
    [{node2, Node} | Config];

%--------------------------
init_per_testcase(benchmark_sender, Config)->
    defaut_sender_init_per_testcase(Config, ?Default_routing_table);

init_per_testcase(benchmark_receiver, Config)->
    defaut_receiver2_init_per_testcase(Config, ?Default_routing_table);
%--------------------------
init_per_testcase(_, Config) ->
    stop_node(?config(node1, Config)),
    stop_node(?config(node2, Config)),
    stop_node(?config(node3, Config)),
    stop_node(?config(node4, Config)),
    Config.
stop_node({Pid, Node}) ->
    case is_node_alive(Node) of
        true ->
            case catch erpc:call(Node, lowpan_node, stop_lowpan_node, [Node, Pid]) of
                ok -> ok;
                {'EXIT', Reason} ->
                    io:format("Error stopping node ~p: ~p~n", [Node, Reason]),
                    {error, stopping_node_failed}
            end;
        false ->
            io:format("Node ~p is already stopped or not reachable.~n", [Node]),
            ok
    end;
stop_node(undefined) ->
    io:format("Node was not started.~n"),
    ok.

is_node_alive(Node) ->
    case catch erpc:call(Node, some_module, ping, []) of
        pong -> true;
        _ -> false
    end.


end_per_testcase(_, _) ->
    ok.

init_per_suite(Config) ->
    Config.
```

```
end_per_suite(_Config) ->
    ok.

%-------------------------------------------------------------------------------
% Send a single payload from node 1 to node 2
%-------------------------------------------------------------------------------
simple_pckt_sender(Config) ->
    {Pid1, Node1} = ?config(node1, Config),
    IPv6Pckt = ?config(ipv6_packet, Config),
    ok = erpc:call(Node1, lowpan_api, sendPacket, [IPv6Pckt, false]),
    ct:pal("Payload sent successfully from node1 to node2"),
    lowpan_node:stop_lowpan_node(Node1, Pid1).

%-------------------------------------------------------------------------------
% Payload from node 1 received by node 2
%-------------------------------------------------------------------------------
simple_pckt_receiver(Config) ->
    {Pid2, Node2} = ?config(node2, Config),
    IPv6Pckt = ?config(ipv6_packet, Config),

    {CompressedHeader, _} = lowpan_core:compressIpv6Header(IPv6Pckt, false),
    PcktInfo = lowpan_core:getPcktInfo(IPv6Pckt),
    Payload = PcktInfo#ipv6PckInfo.payload,
    CompressedIpv6Packet = <<CompressedHeader/binary, Payload/bitstring>>,

    ReceivedData = erpc:call(Node2, lowpan_api, frameReception, []),

    io:format("Expected: ~p~n~nReceived: ~p~n", [CompressedIpv6Packet, ReceivedData]),
    ReceivedData = CompressedIpv6Packet,

    ct:pal("Payload received successfully at node2"),
    lowpan_node:stop_lowpan_node(Node2, Pid2).


%-------------------------------------------------------------------------------
% Send a packet with udp as next header from node 1 to node 2
%-------------------------------------------------------------------------------
simple_udp_pckt_sender(Config) ->
    {Pid1, Node1} = ?config(node1, Config),
    Ipv6Pckt = ?config(ipv6_packet, Config),
    ok = erpc:call(Node1, lowpan_api, sendPacket, [Ipv6Pckt, false]),
    ct:pal("Payload sent successfully from node1 to node2"),
    lowpan_node:stop_lowpan_node(Node1, Pid1).

%-------------------------------------------------------------------------------
% udp pckt reception from node 1 to node 2
%-------------------------------------------------------------------------------
simple_udp_pckt_receiver(Config) ->
    {Pid2, Node2} = ?config(node2, Config),
    IPv6Pckt = ?config(ipv6_packet, Config),

    {CompressedHeader, _} = lowpan_core:compressIpv6Header(IPv6Pckt, false),
    PcktInfo = lowpan_core:getPcktInfo(IPv6Pckt),
    Payload = PcktInfo#ipv6PckInfo.payload,
    CompressedIpv6Packet = <<CompressedHeader/binary, Payload/bitstring>>,

    ReceivedData = erpc:call(Node2, lowpan_api, frameReception, []),

    %io:format("Expected: ~p~n~nReceived: ~p~n", [CompressedIpv6Packet, ReceivedData]),
    ReceivedData = CompressedIpv6Packet,
```

```

    ct:pal("Payload received successfully at node2"),
    lowpan_node:stop_lowpan_node(Node2, Pid2).

%-------------------------------------------------------------------------------
% Send a large payload from node 1 to node 3
%-------------------------------------------------------------------------------
big_payload_sender(Config) ->
    {Pid1, Node1} = ?config(node1, Config),
    IPv6Pckt2 = ?config(ipv6_packet, Config),
    io:format("Size ~p~n",[byte_size(IPv6Pckt2)]),
    ok = erpc:call(Node1, lowpan_api, sendPacket, [IPv6Pckt2, false]),
    ct:pal("Big payload sent successfully from node1 to node3"),
    lowpan_node:stop_lowpan_node(Node1, Pid1).

%-------------------------------------------------------------------------------
% Large payload from node 1 received by node 3
%-------------------------------------------------------------------------------
big_payload_receiver(Config) ->
    {Pid2, Node2}  = ?config(node2, Config),
    IPv6Pckt = ?config(ipv6_packet, Config),

    {CompressedHeader, _} = lowpan_core:compressIpv6Header(IPv6Pckt, false),
    PcktInfo = lowpan_core:getPcktInfo(IPv6Pckt),
    Payload = PcktInfo#ipv6PckInfo.payload,
    CompressedIpv6Packet = <<CompressedHeader/binary, Payload/bitstring>>,

    ReceivedData = erpc:call(Node2, lowpan_api, frameReception, []),

    %io:format("Expected: ~p~n~nReceived: ~p~n", [CompressedIpv6Packet,
        ReceivedData]),
    ReceivedData = CompressedIpv6Packet,

    ct:pal("Big payload received successfully at node2"),
    lowpan_node:stop_lowpan_node(Node2, Pid2).

%-------------------------------------------------------------------------------
% Send packet with a multicast source address
%-------------------------------------------------------------------------------
multicast_sender(Config) ->
    {Pid1, Node1} = ?config(node1, Config),
    {Pid2, Node2} = ?config(node2, Config),

    IPv6Pckt = ?config(ipv6_packet, Config),
    error_multicast_src = erpc:call(Node1, lowpan_api, sendPacket, [IPv6Pckt,
        false]),
    ct:pal("Multicast Source address done"),
    lowpan_node:stop_lowpan_node(Node1, Pid1),
    lowpan_node:stop_lowpan_node(Node2, Pid2).


%-------------------------------------------------------------------------------
% Send packet with the unspecified dest address
%-------------------------------------------------------------------------------
unspecified_dst_sender(Config) ->
    {Pid1, Node1} = ?config(node1, Config),
    {Pid2, Node2} = ?config(node2, Config),

    IPv6Pckt = ?config(ipv6_packet, Config),
    error_unspecified_addr = erpc:call(Node1, lowpan_api, sendPacket, [IPv6Pckt,
        false]),
    ct:pal("Unspecified Source address done"),
```

```
    lowpan_node:stop_lowpan_node(Node1, Pid1),
    lowpan_node:stop_lowpan_node(Node2, Pid2).


%-----------------------------------------------------------------------------
% Send a packet that needs routing from node 1 to node 2
%-----------------------------------------------------------------------------
routing_req_sender(Config) ->
    {Pid1, Node1} = ?config(node1, Config),
    IPv6Pckt = ?config(ipv6_packet, Config),
    erpc:call(Node1, lowpan_api, sendPacket, [IPv6Pckt, false]),
    ct:pal("Routed packet sent successfully from node1 to node2"),
    lowpan_node:stop_lowpan_node(Node1, Pid1).

%-----------------------------------------------------------------------------
% Reception of a routed packet
%-----------------------------------------------------------------------------
routing_req_receiver2(Config) ->
    {Pid2, Node2}  = ?config(node2, Config),
    IPv6Pckt = ?config(ipv6_packet, Config),

    {CompressedHeader, _} = lowpan_core:compressIpv6Header(IPv6Pckt, false),
    PcktInfo = lowpan_core:getPcktInfo(IPv6Pckt),
    Payload = PcktInfo#ipv6PckInfo.payload,
    CompressedIpv6Packet = <<CompressedHeader/binary, Payload/bitstring>>,

    ReceivedData = erpc:call(Node2, lowpan_api, frameReception, []),

    %io:format("Expected: ~p~n~nReceived: ~p~n", [CompressedIpv6Packet,
        ReceivedData]),
    ReceivedData = CompressedIpv6Packet,

    ct:pal("Routed packet received successfully at node2"),
    lowpan_node:stop_lowpan_node(Node2, Pid2).

routing_req_receiver3(Config) ->
    {Pid3, Node3} = ?config(node3, Config),
    erpc:call(Node3, lowpan_api, frameReception, []),
    lowpan_node:stop_lowpan_node(Node3, Pid3).

%-----------------------------------------------------------------------------
% Send a big packet that needs routing from node 1 to node 3
%-----------------------------------------------------------------------------
big_pyld_routing_sender(Config) ->
    {Pid1, Node1} = ?config(node1, Config),
    IPv6Pckt = ?config(ipv6_packet, Config),
    erpc:call(Node1, lowpan_api, sendPacket, [IPv6Pckt, false]),
    ct:pal("Big routed packet sent successfully from node1 to node3"),
    lowpan_node:stop_lowpan_node(Node1, Pid1).

%-----------------------------------------------------------------------------
% Reception of a big payload with routing by node 3
%-----------------------------------------------------------------------------

big_pyld_routing_receiver2(Config) ->
    {Pid2, Node2}  = ?config(node2, Config),
    erpc:call(Node2, lowpan_api, frameReception, []),
    lowpan_node:stop_lowpan_node(Node2, Pid2).

big_pyld_routing_receiver3(Config) ->
    {Pid3, Node3} = ?config(node3, Config),

```

```
    IPv6Pckt = ?config(ipv6_packet, Config),
    {CompressedHeader, _} = lowpan_core:compressIpv6Header(IPv6Pckt, false),
    PcktInfo = lowpan_core:getPcktInfo(IPv6Pckt),
    Payload = PcktInfo#ipv6PckInfo.payload,
    CompressedIpv6Packet = <<CompressedHeader/binary, Payload/bitstring>>,

    ReceivedData = erpc:call(Node3, lowpan_api, frameReception, []),

    %io:format("Expected: ~p~n~nReceived: ~p~n", [CompressedIpv6Packet,
        ReceivedData]),
    ReceivedData = CompressedIpv6Packet,

    ct:pal("Routed packet received successfully at node2"),

    lowpan_node:stop_lowpan_node(Node3, Pid3).


%-------------------------------------------------------------------------------
% Send a datagram with 1 as value for hop left to node 2
%-------------------------------------------------------------------------------
discarded_sender(Config) ->
    {Pid1, Node1} = ?config(node1, Config),
    MeshHeader =
        #mesh_header{
            v_bit = 0,
            f_bit = 0,
            hops_left = 1,
            originator_address =?node1_addr,
            final_destination_address = ?node3_addr
        },

    BinMeshHeader = lowpan_core:buildMeshHeader(MeshHeader),

    Datagram = <<BinMeshHeader/binary, ?Payload/bitstring>>, % meshHeader + Data

    FC = #frame_control{ack_req = ?ENABLED,
                        frame_type = ?FTYPE_DATA,
                        src_addr_mode = ?EXTENDED,
                        dest_addr_mode = ?EXTENDED},
    MacHdr = #mac_header{src_addr =?node1_addr,
                            dest_addr = ?node2_addr},

    ok = erpc:call(Node1, lowpan_api, tx, [Datagram, FC, MacHdr]),
    ct:pal("Packet with 1 hop left sent successfully from node1 to node3"),
    lowpan_node:stop_lowpan_node(Node1, Pid1).

%-------------------------------------------------------------------------------
% Discard datagram received from node 1
%-------------------------------------------------------------------------------
discarded_receiver(Config) ->
    {Pid2, Node2}  = ?config(node2, Config),
    dtg_discarded = erpc:call(Node2, lowpan_api, frameReception, []),
    lowpan_node:stop_lowpan_node(Node2, Pid2).


%-------------------------------------------------------------------------------
% Send a datagram with 0 as value for hop left to node 2
%-------------------------------------------------------------------------------
no_hoplft_dst_reached_sender(Config) ->
    {Pid1, Node1} = ?config(node1, Config),
    MeshHeader =
        #mesh_header{
```

```
            v_bit = 0,
            f_bit = 0,
            hops_left = 0,
            originator_address =?node1_addr,
            final_destination_address = ?node2_addr
        },

    BinMeshHeader = lowpan_core:buildMeshHeader(MeshHeader),

    Datagram = <<BinMeshHeader/binary, ?IPV6_DHTYPE:8, ?Payload/bitstring>>, %
        meshHeader + Data

    FC = #frame_control{ack_req = ?ENABLED,
                        frame_type = ?FTYPE_DATA,
                        src_addr_mode = ?EXTENDED,
                        dest_addr_mode = ?EXTENDED},
    MacHdr = #mac_header{src_addr =?node1_addr,
                            dest_addr = ?node2_addr},

    ok = erpc:call(Node1, lowpan_api, tx, [Datagram, FC, MacHdr]),
    ct:pal("Packet with 0 hop left sent successfully from node1 to node2"),
    lowpan_node:stop_lowpan_node(Node1, Pid1).

%--------------------------------------------------------------------------------
% Reception of datagram with 0 as value for hop left
%--------------------------------------------------------------------------------
no_hoplft_dst_reached_receiver(Config) ->
    {Pid2, Node2}  = ?config(node2, Config),
    Response = erpc:call(Node2, lowpan_api, frameReception, []),
    Response = ?Payload,
    lowpan_node:stop_lowpan_node(Node2, Pid2).

%--------------------------------------------------------------------------------
% Check if error is return when datagram size is unexpected
%--------------------------------------------------------------------------------
unexpected_dtg_size_sender(Config) ->
    {Pid1, Node1}  = ?config(node1, Config),
    IPv6Pckt = ?config(ipv6_packet, Config),
    error_frag_size = erpc:call(Node1, lowpan_api, sendPacket, [IPv6Pckt, false]),
    lowpan_node:stop_lowpan_node(Node1, Pid1).

%--------------------------------------------------------------------------------
% Send payloads from node 1 and node 2 to node 3 with the same tag
%--------------------------------------------------------------------------------
same_tag_different_senders_sender(Config) ->
    {Pid1, Node1} = ?config(node1, Config),
    {Pid2, Node2} = ?config(node2, Config),

    Data1 = <<"Hello ">>,
    Data2 = <<"World!">>,
    PayloadLen = byte_size(Data1) + byte_size(Data2),

    FragHeader1 = #frag_header{
        frag_type = ?FRAG1_DHTYPE,
        datagram_size = PayloadLen,
        datagram_tag = 25,
        datagram_offset = 0
    },
    FragHeader2 = #frag_header{
        frag_type = ?FRAGN_DHTYPE,
        datagram_size = PayloadLen,
        datagram_tag = 25,
```

```
        datagram_offset = 1
    },

    Frag1 = lowpan_core:buildDatagramPckt(FragHeader1, Data1),
    Frag2 = lowpan_core:buildDatagramPckt(FragHeader2, Data2),

    MeshHeader1 =
        #mesh_header{
            v_bit = 0,
            f_bit = 0,
            hops_left = 14,
            originator_address = ?node1_addr,
            final_destination_address = ?node3_addr
        },

    BinMeshHeader1 = lowpan_core:buildMeshHeader(MeshHeader1),

    FC1 = #frame_control{ack_req = ?ENABLED,
                        frame_type = ?FTYPE_DATA,
                        src_addr_mode = ?EXTENDED,
                        dest_addr_mode = ?EXTENDED},
    MH1 = #mac_header{src_addr = ?node1_addr,
                        dest_addr = ?node3_addr},

    MeshHeader2 =
        #mesh_header{
            v_bit = 0,
            f_bit = 0,
            hops_left = 14,
            originator_address = ?node2_addr,
            final_destination_address = ?node3_addr
        },

    BinMeshHeader2 = lowpan_core:buildMeshHeader(MeshHeader2),
    FC2 = #frame_control{ack_req = ?ENABLED,
                        frame_type = ?FTYPE_DATA,
                        src_addr_mode = ?EXTENDED,
                        dest_addr_mode = ?EXTENDED},
    MH2 = #mac_header{src_addr = ?node2_addr,
                        dest_addr = ?node3_addr},

    ok = erpc:call(Node1, lowpan_api, tx, [<<BinMeshHeader1/binary, Frag1/
        bitstring>>, FC1, MH1]),
    ok = erpc:call(Node2, lowpan_api, tx, [<<BinMeshHeader2/binary, Frag1/
        bitstring>>, FC2, MH2]),

    ok = erpc:call(Node1, lowpan_api, tx, [<<BinMeshHeader1/binary, Frag2/
        bitstring>>, FC1, MH1]),
    ok = erpc:call(Node2, lowpan_api, tx, [<<BinMeshHeader2/binary, Frag2/
        bitstring>>, FC2, MH2]),

    ct:pal("Fragments sent from node1 and node2 to node3 with the same tag"),
    lowpan_node:stop_lowpan_node(Node1, Pid1),
    lowpan_node:stop_lowpan_node(Node2, Pid2).

%-------------------------------------------------------------------------------
% Reception of payloads from node 1 and node 2 by node 3 with the same tag
%-------------------------------------------------------------------------------
same_tag_different_senders_receiver(Config) ->
    {Pid3, Node3} = ?config(node3, Config),

    % Receive and reassemble the fragments
```

```
    ReceivedData1 = erpc:call(Node3, lowpan_api, frameReception, []),
    ReceivedData2 = erpc:call(Node3, lowpan_api, frameReception, []),

    ExpectedData = <<"Hello World!">>,
    %io:format("Expected: ~p~n~nReceived 1: ~p~n~nReceived 2: ~p~n", [ExpectedData
        , ReceivedData1, ReceivedData2]),

    case (ReceivedData1 == ExpectedData) andalso (ReceivedData2 == ExpectedData)
        of
        true ->
            ct:pal("Payloads received successfully at node3 with the same tag from
                different senders"),
            lowpan_node:stop_lowpan_node(Node3, Pid3);
        false ->
            ct:fail("Payloads did not match expected data"),
            lowpan_node:stop_lowpan_node(Node3, Pid3)
    end.


%-------------------------------------------------------------------------------
% Send incomplete payload from node 1 to node 2 to trigger a timeout
%-------------------------------------------------------------------------------
timeout_sender(Config) ->
    {Pid1, Node1} = ?config(node1, Config),

    Data = <<"Hello World!">>,
    PayloadLen = byte_size(Data),

    FragHeader1 = #frag_header{
        frag_type = ?FRAG1_DHTYPE,
        datagram_size = PayloadLen,
        datagram_tag = 25,
        datagram_offset = 0
    },

    Frag1 = lowpan_core:buildDatagramPckt(FragHeader1, <<"Hello ">>),
     MeshHeader1 =
        #mesh_header{
            v_bit = 0,
            f_bit = 0,
            hops_left = 14,
            originator_address = ?node1_addr,
            final_destination_address = ?node2_addr
        },

    BinMeshHeader1 = lowpan_core:buildMeshHeader(MeshHeader1),

    FC1 = #frame_control{ack_req = ?ENABLED,
                        frame_type = ?FTYPE_DATA,
                        src_addr_mode = ?EXTENDED,
                        dest_addr_mode = ?EXTENDED},
    MH1 = #mac_header{src_addr = ?node1_addr,
                        dest_addr = ?node2_addr},



    ok = erpc:call(Node1, lowpan_api, tx, [<<BinMeshHeader1/binary, Frag1/
        bitstring>>, FC1, MH1]),

    ct:pal("Incomplete payload sent from node1 to node2 to trigger a timeout"),
    lowpan_node:stop_lowpan_node(Node1, Pid1).

```

```
%-------------------------------------------------------------------------------
% Receiver node 2 should experience a timeout
%-------------------------------------------------------------------------------
timeout_receiver(Config) ->
    {Pid2, Node2} = ?config(node2, Config),
    reassembly_timeout = erpc:call(Node2, lowpan_api, frameReception, []),
    ct:pal("Timeout occurred~n"),
    lowpan_node:stop_lowpan_node(Node2, Pid2).

%-------------------------------------------------------------------------------
% Send duplicate fragment to node 2
%-------------------------------------------------------------------------------
duplicate_sender(Config) ->
    {Pid1, Node1} = ?config(node1, Config),

    Data1 = <<"Hello ">>,
    Data2 = <<"World!">>,
    PayloadLen = byte_size(Data1) + byte_size(Data2),

    FragHeader1 = #frag_header{
        frag_type = ?FRAG1_DHTYPE,
        datagram_size = PayloadLen,
        datagram_tag = 25,
        datagram_offset = 0
    },
    FragHeader2 = #frag_header{
        frag_type = ?FRAGN_DHTYPE,
        datagram_size = PayloadLen,
        datagram_tag = 25,
        datagram_offset = 1
    },

    Frag1 = lowpan_core:buildDatagramPckt(FragHeader1, Data1),
    Frag2 = lowpan_core:buildDatagramPckt(FragHeader2, Data2),

    MeshHeader =
        #mesh_header{
            v_bit = 0,
            f_bit = 0,
            hops_left = 14,
            originator_address = ?node1_addr,
            final_destination_address = ?node2_addr
        },

    BinMeshHeader = lowpan_core:buildMeshHeader(MeshHeader),

    FC = #frame_control{ack_req = ?ENABLED,
                        frame_type = ?FTYPE_DATA,
                        src_addr_mode = ?EXTENDED,
                        dest_addr_mode = ?EXTENDED},
    MH = #mac_header{src_addr = ?node1_addr,
                        dest_addr = ?node2_addr},

    ok = erpc:call(Node1, lowpan_api, tx, [<<BinMeshHeader/binary, Frag1/bitstring
        >>, FC, MH]),
    ok = erpc:call(Node1, lowpan_api, tx, [<<BinMeshHeader/binary, Frag1/bitstring
        >>, FC, MH]), % duplicated fragment
    ok = erpc:call(Node1, lowpan_api, tx, [<<BinMeshHeader/binary, Frag2/bitstring
        >>, FC, MH]),

    ct:pal("Fragments sent from node1 and node2 to node3 with the same tag"),
    lowpan_node:stop_lowpan_node(Node1, Pid1).
```

```
%------------------------------------------------------------------------------
% Reception of payloads from node 1 and node 2 by node 3 with the same tag
%------------------------------------------------------------------------------
duplicate_receiver(Config) ->
    {Pid2, Node2} = ?config(node2, Config),

    ReceivedData1 = erpc:call(Node2, lowpan_api, frameReception, []),

    ExpectedData = <<"Hello World!">>,
    %io:format("Expected: ~p~n~nReceived: ~p~n", [ExpectedData, ReceivedData1]),
    ReceivedData1 = ExpectedData,
    lowpan_node:stop_lowpan_node(Node2, Pid2).


%------------------------------------------------------------------------------
% Send a packet that needs routing from node 1 to node 4
%------------------------------------------------------------------------------
multiple_hop_sender(Config) ->
    {Pid1, Node1} = ?config(node1, Config),
    IPv6Pckt = ?config(ipv6_packet, Config),
    ok = erpc:call(Node1, lowpan_api, sendPacket, [IPv6Pckt, false]),
    ct:pal("multi hop packet sent successfully from node1 to node4"),
    lowpan_node:stop_lowpan_node(Node1, Pid1).

%------------------------------------------------------------------------------
% Reception of a routed packet
%------------------------------------------------------------------------------
multiple_hop_receiver2(Config) ->
    {Pid2, Node2} = ?config(node2, Config),
    erpc:call(Node2, lowpan_api, frameReception, []),
    lowpan_node:stop_lowpan_node(Node2, Pid2).


multiple_hop_receiver3(Config) ->
    {Pid3, Node3} = ?config(node3, Config),
    erpc:call(Node3, lowpan_api, frameReception, []),
    lowpan_node:stop_lowpan_node(Node3, Pid3).

multiple_hop_receiver4(Config) ->
    {Pid4, Node4}  = ?config(node4, Config),
    IPv6Pckt = ?config(ipv6_packet, Config),

    {CompressedHeader, _} = lowpan_core:compressIpv6Header(IPv6Pckt, true),
    PcktInfo = lowpan_core:getPcktInfo(IPv6Pckt),
    Payload = PcktInfo#ipv6PckInfo.payload,
    CompressedIpv6Packet = <<CompressedHeader/binary, Payload/bitstring>>,

    ReceivedData = erpc:call(Node4, lowpan_api, frameReception, []),

    io:format("Expected: ~p~n~nReceived: ~p~n", [CompressedIpv6Packet,
        ReceivedData]),
    ReceivedData = CompressedIpv6Packet,

    ct:pal("Routed packet received successfully at node4"),
    lowpan_node:stop_lowpan_node(Node4, Pid4).


%------------------------------------------------------------------------------
% Send a none lowpan frame to node 2
%------------------------------------------------------------------------------
nalp_sender(Config) ->
```

```
    {Pid1, Node1} = ?config(node1, Config),
    IPv6Pckt = ?config(ipv6_packet, Config),

    Frame = <<?NALP_DHTYPE, IPv6Pckt/bitstring>>,

    FC = #frame_control{ack_req = ?ENABLED,
                        frame_type = ?FTYPE_DATA,
                        src_addr_mode = ?EXTENDED,
                        dest_addr_mode = ?EXTENDED},

    MH = #mac_header{src_addr = ?node1_addr,
        dest_addr = ?node2_addr},

    error_nalp = erpc:call(Node1, lowpan_api, tx, [Frame, FC, MH]),
    ct:pal("NALP error correctly received"),
    lowpan_node:stop_lowpan_node(Node1, Pid1).


%-------------------------------------------------------------------------------
% Send a broadcast packet
%-------------------------------------------------------------------------------
broadcast_sender(Config) ->
    {Pid1, Node1} = ?config(node1, Config),
    IPv6Pckt = ?config(ipv6_packet, Config),
    ok = erpc:call(Node1, lowpan_api, sendPacket, [IPv6Pckt, false]),
    ct:pal("Broadcast packet sent successfully"),
    lowpan_node:stop_lowpan_node(Node1, Pid1).

%-------------------------------------------------------------------------------
% Reception of a broadcasted packet
%-------------------------------------------------------------------------------
broadcast_receiver(Config) ->
    {Pid2, Node2} = ?config(broadcast_node, Config),
    IPv6Pckt = ?config(ipv6_packet, Config),

    {CompressedHeader, _} = lowpan_core:compressIpv6Header(IPv6Pckt, false),
    PcktInfo = lowpan_core:getPcktInfo(IPv6Pckt),
    Payload = PcktInfo#ipv6PckInfo.payload,
    CompressedIpv6Packet = <<CompressedHeader/binary, Payload/bitstring>>,

    ReceivedData = erpc:call(Node2, lowpan_api, frameReception, []),

    %io:format("Expected: ~p~n~nReceived: ~p~n", [CompressedIpv6Packet,
        ReceivedData]),
    ReceivedData = CompressedIpv6Packet,

    ct:pal("Routed packet received successfully at node4"),

    lowpan_node:stop_lowpan_node(Node2, Pid2).


%-------------------------------------------------------------------------------
% Send a datagram with special hopsleft value 0xF
%-------------------------------------------------------------------------------
extended_hopsleft_sender(Config) ->
    {Pid1, Node1} = ?config(node1, Config),
    IPv6Pckt = ?config(ipv6_packet, Config),
    ok = erpc:call(Node1, lowpan_api, extendedHopsleftTx, [IPv6Pckt]),
    ct:pal("extended hop left packet sent successfully from node1 to node4"),
    lowpan_node:stop_lowpan_node(Node1, Pid1).

%-------------------------------------------------------------------------------
```

```
% Discard datagram received from node 1
%----------------------------------------------------------------------------
extended_hopsleft_receiver2(Config) ->
    {Pid2, Node2} = ?config(node2, Config),
    erpc:call(Node2, lowpan_api, frameReception, []),
    lowpan_node:stop_lowpan_node(Node2, Pid2).


extended_hopsleft_receiver3(Config) ->
    {Pid3, Node3} = ?config(node3, Config),
    erpc:call(Node3, lowpan_api, frameReception, []),
    lowpan_node:stop_lowpan_node(Node3, Pid3).

extended_hopsleft_receiver4(Config) ->
    {Pid4, Node4}  = ?config(node4, Config),
    IPv6Pckt = ?config(ipv6_packet, Config),

    {CompressedHeader, _} = lowpan_core:compressIpv6Header(IPv6Pckt, true),
    PcktInfo = lowpan_core:getPcktInfo(IPv6Pckt),
    Payload = PcktInfo#ipv6PckInfo.payload,
    CompressedIpv6Packet = <<CompressedHeader/binary, Payload/bitstring>>,

    ReceivedData = erpc:call(Node4, lowpan_api, frameReception, []),

    %io:format("Expected: ~p~n~nReceived: ~p~n", [CompressedIpv6Packet,
        ReceivedData]),
    ReceivedData = CompressedIpv6Packet,

    ct:pal("Routed packet received successfully at node4"),
    lowpan_node:stop_lowpan_node(Node4, Pid4).

%----------------------------------------------------------------------------
% Send a packet in mesh level scope (mesh-local prefix used)
%----------------------------------------------------------------------------
mesh_prefix_sender(Config) ->
    {Pid1, Node1} = ?config(node1, Config),
    IPv6Pckt = ?config(ipv6_packet, Config),
    ok = erpc:call(Node1, lowpan_api, sendPacket, [IPv6Pckt, false]),
    ct:pal("Broadcast packet sent successfully"),
    lowpan_node:stop_lowpan_node(Node1, Pid1).

%----------------------------------------------------------------------------
% Reception of a  packet
%----------------------------------------------------------------------------
mesh_prefix_receiver(Config) ->
    {Pid2, Node2} = ?config(node2, Config),
    IPv6Pckt = ?config(ipv6_packet, Config),

    {CompressedHeader, _} = lowpan_core:compressIpv6Header(IPv6Pckt, false),
    PcktInfo = lowpan_core:getPcktInfo(IPv6Pckt),
    Payload = PcktInfo#ipv6PckInfo.payload,
    CompressedIpv6Packet = <<CompressedHeader/binary, Payload/bitstring>>,

    ReceivedData = erpc:call(Node2, lowpan_api, frameReception, []),

    %%io:format("Expected: ~p~n~nReceived: ~p~n", [CompressedIpv6Packet,
        ReceivedData]),
    ReceivedData = CompressedIpv6Packet,

    ct:pal("Routed packet received successfully at node4"),

    lowpan_node:stop_lowpan_node(Node2, Pid2).
```

A.11 Robot application code

```
-module(robot).

-behaviour(application).

-include("utils.hrl").

-export([
    tx/0,
    tx3/0,
    tx4/0,
    tx5/0,
    tx_unc_ipv6/0,
    tx_iphc_pckt/0,
    tx_frag_iphc_pckt/0,
    tx_big_payload/1,
    tx_with_udp/0,
    tx_msh_iphc_pckt/0,
    tx_msh_frag_iphc_pckt/0,
    msh_pckt_tx/0,
    msh_big_pckt_tx/0,
    rx/0,
    tx_broadcast_pckt/0,
    extendedHopsleftTx/0,
    tx_unc_ipv6_udp/0,
    tx_comp_ipv6_udp/0,
    tx_mesh_prefix/0,
    ieeetx2/0,
    ieeetx3/0,
    tx_big_payload3/1,
    tx_big_payload4/1,
    tx_big_payload5/1
]).

-export([start/2]).
-export([stop/1]).


%--- Macros ---------------------------------------------------------------

-define(TX_ANTD, 16450).
-define(RX_ANTD, 16450).

%-------------------------------------------------------------------------
% Sends uncompressed ipv6 packet format
%-------------------------------------------------------------------------
tx_unc_ipv6() ->
    Ipv6Pckt = ipv6:buildIpv6Packet(?IPv6Header, ?Payload),
    io:format("Frame ~p~n", [Ipv6Pckt]),
    io:format("Fragment size: ~p bytes~n", [byte_size(Ipv6Pckt)]),

    lowpan_api:sendUncDatagram(Ipv6Pckt, ?FrameControl, ?MacHeader).


%-------------------------------------------------------------------------
```

```
% Sends compressed header packet format
%----------------------------------------------------------------------------
tx_iphc_pckt() ->
    InlineData = <<12:8, ?Node1MacAddress/binary, ?Node2MacAddress/binary>>,
    ExpectedHeader =
        <<?IPHC_DHTYPE:3, 3:2, 12:1, 3:2, 0:1, 0:1, 1:2, 0:1, 0:1, 1:2, InlineData
            /binary>>,

    % Create the IPHC packet
    IPHC = lowpan_core:createIphcPckt(ExpectedHeader, ?Payload),
    io:format("IphcHeader ~p~n", [IPHC]),
    io:format("Fragment size: ~p bytes~n", [byte_size(IPHC)]),

    lowpan_api:tx(IPHC, ?FrameControl, ?MacHeader).

%----------------------------------------------------------------------------
% Sends meshed and compressed header packet format
%----------------------------------------------------------------------------
tx_msh_iphc_pckt() ->
    Ipv6Pckt = ipv6:buildIpv6Packet(?IPv6Header, ?Payload),
    {CompressedHeader, _} = lowpan_core:compressIpv6Header(Ipv6Pckt, true),

    MeshHeader =
        #mesh_header{
            v_bit = 0,
            f_bit = 0,
            hops_left = 14,
            originator_address = ?Node1MacAddress,
            final_destination_address = ?Node2MacAddress
        },

    BinMeshHeader = lowpan_core:buildMeshHeader(MeshHeader),
    Datagram = <<BinMeshHeader/binary, CompressedHeader/binary, ?Payload/bitstring
        >>,
    io:format("Datagram ~p~n", [Datagram]),

    lowpan_api:tx(Datagram, ?FrameControl, ?MacHeader).

%----------------------------------------------------------------------------
% Sends fragmented and compressed packet format
%----------------------------------------------------------------------------
tx_frag_iphc_pckt() ->
    Ipv6Pckt = ipv6:buildIpv6Packet(?IPv6Header, ?Payload),
    {CompressedHeader, _} = lowpan_core:compressIpv6Header(Ipv6Pckt, false),
    PcktLen = byte_size(Ipv6Pckt),

    FragHeader =
        #frag_header{
            frag_type = ?FRAG1_DHTYPE,
            datagram_size = PcktLen,
            datagram_tag = 124
        },

    FragHeaderBin = lowpan_core:buildFirstFragHeader(FragHeader),

    Datagram = <<FragHeaderBin/binary, CompressedHeader/binary, ?Payload/bitstring
        >>,
    io:format("Frame ~p~n", [Datagram]),
    io:format("Fragment size: ~p bytes~n", [byte_size(Datagram)]),

    lowpan_api:tx(Datagram, ?FrameControl, ?MacHeader).

```

```
%-------------------------------------------------------------------------------
% Sends meshed, fragmented and compressed packet format
%-------------------------------------------------------------------------------
tx_msh_frag_iphc_pckt() ->
    Ipv6Pckt = ipv6:buildIpv6Packet(?IPv6Header, ?Payload),
    {CompressedHeader, _} = lowpan_core:compressIpv6Header(Ipv6Pckt, true),
    PcktLen = byte_size(Ipv6Pckt),

    FragHeader =
        #frag_header{
            frag_type = ?FRAG1_DHTYPE,
            datagram_size = PcktLen,
            datagram_tag = 124
        },

    FragHeaderBin = lowpan_core:buildFirstFragHeader(FragHeader),

    MeshHeader =
        #mesh_header{
            v_bit = 0,
            f_bit = 0,
            hops_left = 14,
            originator_address = ?Node1MacAddress,
            final_destination_address = ?Node2MacAddress
        },

    BinMeshHeader = lowpan_core:buildMeshHeader(MeshHeader),
    Datagram =
        <<BinMeshHeader/binary, FragHeaderBin/binary, CompressedHeader/binary, ?
            Payload/bitstring>>,
    io:format("Datagram ~p~n", [Datagram]),

    lowpan_api:tx(Datagram, ?FrameControl, ?MacHeader).

%-------------------------------------------------------------------------------
% Sends broadcast packet format
%-------------------------------------------------------------------------------
tx_broadcast_pckt() ->
    Ipv6Pckt = ipv6:buildIpv6Packet(?IPv6Header, ?Payload),
    {CompressedHeader, _} = lowpan_core:compressIpv6Header(Ipv6Pckt, false),
    PcktLen = byte_size(Ipv6Pckt),

    FragHeader =
        #frag_header{
            frag_type = ?FRAG1_DHTYPE,
            datagram_size = PcktLen,
            datagram_tag = 124
        },

    FragHeaderBin = lowpan_core:buildFirstFragHeader(FragHeader),

    DestMacAddr = lowpan_core:generateEUI64MacAddr(<<16#1234:16>>),

    DestAddr = <<16#FF02:16, 0:64, 1:16, 16#FF00:16, 16#1234:16>>,
    DestAddress = binary:decode_unsigned(DestAddr),
    {_, BroadcastHeader, _} = lowpan_core:getNextHop(?Node1MacAddress, ?
        Node1MacAddress, DestMacAddr, DestAddress, 3, false),

    Datagram =
        <<BroadcastHeader/binary, FragHeaderBin/binary, CompressedHeader/binary, ?
            Payload/bitstring>>,
    io:format("Datagram ~p~n", [Datagram]),
```

```

    MacHeader = #mac_header{src_addr = ?Node1MacAddress, dest_addr = DestMacAddr},
    lowpan_api:tx(Datagram, ?FrameControl, MacHeader).


%----------------------------------------------------------------------------
% Simple transmission to node 2
%----------------------------------------------------------------------------
tx() ->
    Ipv6Pckt = ipv6:buildIpv6Packet(?IPv6Header, ?Payload),
    lowpan_api:sendPacket(Ipv6Pckt, true).

%----------------------------------------------------------------------------
% Simple transmission to node 3
%----------------------------------------------------------------------------
tx3() ->
    Ipv6Pckt = ipv6:buildIpv6Packet(?IPv6Header3, ?Payload),
    lowpan_api:sendPacket(Ipv6Pckt, true).

%----------------------------------------------------------------------------
% Simple transmission to node 4
%----------------------------------------------------------------------------
tx4() ->
    Ipv6Pckt = ipv6:buildIpv6Packet(?IPv6Header4, ?Payload),
    lowpan_api:sendPacket(Ipv6Pckt, true).

%----------------------------------------------------------------------------
% Simple transmission to node 5
%----------------------------------------------------------------------------
tx5() ->
    Ipv6Pckt = ipv6:buildIpv6Packet(?IPv6Header5, ?Payload),
    lowpan_api:sendPacket(Ipv6Pckt, true).



%----------------------------------------------------------------------------
% Big payload transmission, N = nbr of chunck in payload
%----------------------------------------------------------------------------
tx_big_payload(N) ->
    Payload = lowpan_core:generateChunks(N),

    Node1Address = lowpan_core:generateLLAddr(?Node1MacAddress),
    Node2Address = lowpan_core:generateLLAddr(?Node2MacAddress),
    PayloadLength = byte_size(Payload),

    IPv6Header =
        #ipv6_header{
            version = 6,
            traffic_class = 0,
            flow_label = 0,
            payload_length = PayloadLength,
            next_header = 12,
            hop_limit = 64,
            source_address = Node1Address,
            destination_address = Node2Address
        },
    Ipv6Pckt = ipv6:buildIpv6Packet(IPv6Header, Payload),
    lowpan_api:sendPacket(Ipv6Pckt, true).

%% Tx to node 3
tx_big_payload3(N) ->
    Payload = lowpan_core:generateChunks(N),
```

```

    Node1Address = lowpan_core:generateLLAddr(?Node1MacAddress),
    Node2Address = lowpan_core:generateLLAddr(?Node3MacAddress),
    PayloadLength = byte_size(Payload),

    IPv6Header =
        #ipv6_header{
            version = 6,
            traffic_class = 0,
            flow_label = 0,
            payload_length = PayloadLength,
            next_header = 12,
            hop_limit = 64,
            source_address = Node1Address,
            destination_address = Node2Address
        },
    Ipv6Pckt = ipv6:buildIpv6Packet(IPv6Header, Payload),
    lowpan_api:sendPacket(Ipv6Pckt, true).

%% Tx to node 4
tx_big_payload4(N) ->
    Payload = lowpan_core:generateChunks(N),

    Node1Address = lowpan_core:generateLLAddr(?Node1MacAddress),
    Node2Address = lowpan_core:generateLLAddr(?Node4MacAddress),
    PayloadLength = byte_size(Payload),

    IPv6Header =
        #ipv6_header{
            version = 6,
            traffic_class = 0,
            flow_label = 0,
            payload_length = PayloadLength,
            next_header = 12,
            hop_limit = 64,
            source_address = Node1Address,
            destination_address = Node2Address
        },
    Ipv6Pckt = ipv6:buildIpv6Packet(IPv6Header, Payload),
    lowpan_api:sendPacket(Ipv6Pckt, true).

%% Tx to node 5
tx_big_payload5(N) ->
    Payload = lowpan_core:generateChunks(N),

    Node1Address = lowpan_core:generateLLAddr(?Node1MacAddress),
    Node2Address = lowpan_core:generateLLAddr(?Node5MacAddress),
    PayloadLength = byte_size(Payload),

    IPv6Header =
        #ipv6_header{
            version = 6,
            traffic_class = 0,
            flow_label = 0,
            payload_length = PayloadLength,
            next_header = 12,
            hop_limit = 64,
            source_address = Node1Address,
            destination_address = Node2Address
        },
    Ipv6Pckt = ipv6:buildIpv6Packet(IPv6Header, Payload),
    lowpan_api:sendPacket(Ipv6Pckt, true).
```

```


%------------------------------------------------------------------------------
% Transmission of uncompressed ipv6 packet with udp next header
%------------------------------------------------------------------------------
tx_unc_ipv6_udp() ->
    Payload = <<"Hello world">>,
    PayloadLength = byte_size(Payload),
    IPv6Header =
        #ipv6_header{
            version = 6,
            traffic_class = 0,
            flow_label = 0,
            % 4 bytes for the UDP header
            payload_length = PayloadLength,
            next_header = 17,
            hop_limit = 255,
            source_address = ?Node1Address,
            destination_address = ?Node2Address
        },
    UdpHeader =
        #udp_header{
            source_port = 1025,
            destination_port = 61617,
            length = PayloadLength,
            checksum = 16#f88c
        },

    Ipv6Pckt = ipv6:buildIpv6UdpPacket(IPv6Header, UdpHeader, Payload),
    io:format("Frame ~p~n", [Ipv6Pckt]),
    io:format("Fragment size: ~p bytes~n", [byte_size(Ipv6Pckt)]),

    lowpan_api:sendUncDatagram(Ipv6Pckt, ?FrameControl, ?MacHeader).


%------------------------------------------------------------------------------
% Transmission of compressed ipv6 packet with udp next header
%------------------------------------------------------------------------------
tx_comp_ipv6_udp() ->
    Payload = <<"Hello world">>,
    PayloadLength = byte_size(Payload),
    IPv6Header =
        #ipv6_header{
            version = 6,
            traffic_class = 0,
            flow_label = 0,
            % 4 bytes for the UDP header
            payload_length = PayloadLength,
            next_header = 17,
            hop_limit = 64,
            source_address = ?Node1Address,
            destination_address = ?Node2Address
        },
    UdpHeader =
        #udp_header{
            source_port = 1025,
            destination_port = 61617,
            length = PayloadLength,
            checksum = 16#f88c
        },

    Ipv6Pckt = ipv6:buildIpv6UdpPacket(IPv6Header, UdpHeader, Payload),
```

```
    lowpan_api:sendPacket(Ipv6Pckt).

%-----------------------------------------------------------------------------
% Ipv6 with nextHeader packet format verification
%-----------------------------------------------------------------------------
tx_with_udp() ->
    IPv6Header =
        #ipv6_header{
            version = 6,
            traffic_class = 0,
            flow_label = 0,
            % 4 bytes for the UDP header
            payload_length = ?PayloadLength,
            next_header = 17,
            hop_limit = 64,
            source_address = ?Node1Address,
            destination_address = ?Node2Address
        },
    UdpHeader =
        #udp_header{
            source_port = 1025,
            destination_port = 61617,
            length = ?PayloadLength,
            checksum = 16#f88c
        },

    Ipv6Pckt = ipv6:buildIpv6UdpPacket(IPv6Header, UdpHeader, ?Payload),
    lowpan_api:sendPacket(Ipv6Pckt).

%-----------------------------------------------------------------------------
% Transmission of packet that needs routing
%-----------------------------------------------------------------------------
msh_pckt_tx() ->
    IPv6Header =
        #ipv6_header{
            version = 6,
            traffic_class = 0,
            flow_label = 0,
            payload_length = ?PayloadLength,
            next_header = 10,
            hop_limit = 64,
            source_address = ?Node1Address,
            destination_address = ?Node3Address
        },

    Ipv6Pckt = ipv6:buildIpv6Packet(IPv6Header, ?Payload),
    lowpan_api:sendPacket(Ipv6Pckt).

%-----------------------------------------------------------------------------
% Transmission of big packet that needs routing
%-----------------------------------------------------------------------------
msh_big_pckt_tx() ->
    IPv6Header =
        #ipv6_header{
            version = 6,
            traffic_class = 0,
            flow_label = 0,
            payload_length = ?PayloadLength,
            next_header = 10,
            hop_limit = 64,
            source_address = ?Node1Address,
            destination_address = ?Node3Address
```

```
        },

    Ipv6Pckt = ipv6:buildIpv6Packet(IPv6Header, ?BigPayload),
    lowpan_api:sendPacket(Ipv6Pckt).

%--------------------------------------------------------------------------------
% Extended hopsLeft packet transmission
%--------------------------------------------------------------------------------
extendedHopsleftTx() ->
    Ipv6Pckt = ipv6:buildIpv6Packet(?IPv6Header, ?Payload),
    lowpan_api:extendedHopsleftTx(Ipv6Pckt).

%--------------------------------------------------------------------------------
% Transmission of mesh level packet (mesh-local prefix used)
%--------------------------------------------------------------------------------
tx_mesh_prefix() ->
    MacAddress = lowpan_core:generateEUI64MacAddr(?Node2MacAddress),
    IPv6Header = #ipv6_header{
            version = 6,
            traffic_class = 0,
            flow_label = 0,
            payload_length = byte_size(?Payload),
            next_header = 12,
            hop_limit = 64,
            source_address = ?Node1Address,
            destination_address =  <<?MESH_LOCAL_PREFIX:16, 16#0DB8:16, 0:32,
                2:8,0:48, MacAddress/binary>>
        },
    Packet = ipv6:buildIpv6Packet(IPv6Header, ?Payload),
    lowpan_api:sendPacket(Packet).

%--------------------------------------------------------------------------------
% Data reception
%--------------------------------------------------------------------------------
rx() ->
    grisp_led:color(2, red),
    lowpan_api:frameReception(),
    grisp_led:color(2, green),
    rx().

%--------------------------------------------------------------------------------
% Direct ieee data transmission to node 2
%--------------------------------------------------------------------------------
ieeetx2()->
    FrameControl = #frame_control{
    frame_type = ?FTYPE_DATA,
    src_addr_mode = ?EXTENDED,
    dest_addr_mode = ?EXTENDED},

    MacHeader = #mac_header{src_addr = ?Node1MacAddress,
                dest_addr = ?Node2MacAddress},

    lowpan_api:tx(<<"Hello">>, FrameControl, MacHeader).

%--------------------------------------------------------------------------------
% Direct ieee data transmission to node 3
%--------------------------------------------------------------------------------
ieeetx3()->
    FrameControl = #frame_control{
    frame_type = ?FTYPE_DATA,
    src_addr_mode = ?EXTENDED,
    dest_addr_mode = ?EXTENDED},
```

```erlang

    MacHeader = #mac_header{src_addr = ?Node1MacAddress,
                dest_addr = ?Node3MacAddress},

    lowpan_api:tx(<<"Hello">>, FrameControl, MacHeader).


%-------------------------------------------------------------------------------
% IEEE 802.15.4 setup only for manual configuration
%-------------------------------------------------------------------------------
% ieee802154_setup(MacAddr)->
%     ieee802154:start(#ieee_parameters{
%         duty_cycle = duty_cycle_non_beacon,
%         input_callback = fun lowpan_api:input_callback/4
%     }),

%     case application:get_env(robot, pan_id) of
%         {ok, PanId} ->
%             ieee802154:set_pib_attribute(mac_pan_id, PanId);
%         _ ->
%             ok
%     end,

%     case byte_size(MacAddr) of
%         ?EXTENDED_ADDR_LEN -> ieee802154:set_pib_attribute(mac_extended_address, MacAddr);
%         ?SHORT_ADDR_LEN -> ieee802154:set_pib_attribute(mac_short_address, MacAddr)
%     end,

%     ieee802154:rx_on().

start(_Type, _Args) ->
    {ok, Supervisor} = robot_sup:start_link(),
    grisp:add_device(spi2, pmod_uwb),
    pmod_uwb:write(tx_antd, #{tx_antd => ?TX_ANTD}),
    pmod_uwb:write(lde_if, #{lde_rxantd => ?RX_ANTD}),

    NodeMacAddr = case application:get_env(robot, mac_addr) of
        {ok, MacAddr} ->
            MacAddr;
        _ ->
            ?Node1MacAddress
    end,

    %ieee802154_setup(NodeMacAddr),

    lowpan_api:start(#{node_mac_addr => NodeMacAddr, routing_table => ?Default_routing_table}),

    %rx(),
    {ok, Supervisor}.

% @private
stop(_State) ->
    ok.
```

A.12 Mockups files code

```
-module(mock_mac).

-include("../src/mac_frame.hrl").

-include_lib("eunit/include/eunit.hrl").

-behaviour(gen_mac_layer).

-export([send_data/3]).
% -export([send_data/4]).

% -export([reception/1]).
-export([reception/0]).
%%% gen_server callbacks
-export([init/1]).
-export([tx/4]).
-export([rx/1]).
-export([rx_on/2]).
-export([rx_off/1]).
-export([set/3]).
-export([get/2]).
-export([terminate/2]).

% --- API ------------------
reception() ->
    mac_layer_behaviour:rx().

send_data(FrameControl, MacHeader, Payload) ->
    mac_layer_behaviour:tx(FrameControl, MacHeader, Payload).

% --- Callbacks -----------
init(Params) ->
    PhyModule =
        case Params of
            #{phy_layer := PHY} ->
                PHY;
            _ ->
                pmod_uwb
        end,
    #{
        phy_layer => PhyModule,
        rx => off,
        pib =>
            #{
                mac_pan_id => <<16#FFFF:16>>,
                mac_extended_address => <<16#FFFFFFFF:64>>,
                mac_short_address => <<16#FFFF:16>>
            }
    }.

tx(
    State,
    #frame_control{ack_req = ?ENABLED} = FrameControl,
    #mac_header{seqnum = Seqnum} = MacHeader,
    Payload
) ->
    Frame = mac_frame:encode(FrameControl, MacHeader, Payload),
    transmission(Frame),
```

```
    _RxFrame = receive_ack(Seqnum),
    {ok, State};
tx(State, FrameControl, MacHeader, Payload) ->
    Frame = mac_frame:encode(FrameControl, MacHeader, Payload),
    {transmission(Frame), State}.

rx(State) ->
    {ok, State, receive_()}.

rx_on(State, _) ->
    {ok, State#{rx => on}}.

rx_off(State) ->
    {ok, State#{rx => off}}.

get(#{pib := PIB} = State, Attribute) ->
    case maps:get(Attribute, PIB) of
        {badkey, _} ->
            {error, State, unsupported_attribute};
        Val ->
            {ok, State, Val}
    end.

set(#{pib := PIB} = State, Attribute, Value) ->
    case maps:update(Attribute, Value, PIB) of
        {badkey, _} ->
            {error, State, unsupported_attribute};
        NewPIB ->
            {ok, State#{pib => NewPIB}}
    end.

terminate(_State, _Reason) ->
    ok.

% --- Internals ---------

receive_() ->
    FrameControl = #frame_control{pan_id_compr = ?ENABLED, frame_version = 2#00},
    MacHeader =
        #mac_header{
            seqnum = 0,
            dest_pan = <<16#DECA:16>>,
            dest_addr = <<"RX">>,
            src_pan = <<16#DECA:16>>,
            src_addr = <<"TX">>
        },
    {FrameControl, MacHeader, <<"Hello">>}.

% Received MAC frame for an ACK is only composed of the Frame control, the seqnum
    and the FCS
receive_ack(Seqnum) ->
    FrameControl = #frame_control{frame_type = ?FTYPE_ACK},
    MacHeader = #mac_header{seqnum = Seqnum},
    {FrameControl, MacHeader, <<>>}.

transmission(Frame) when byte_size(Frame) < 125 ->
    io:format("~w~n", [Frame]),
    ok.
```

```
-module(mock_phy_network).
-behaviour(gen_statem).
```

```erlang

-include("../src/mac_frame.hrl").


-export([start_link/2]).
-export([start/2]).
-export([stop_link/0]).

-export([transmit/2]).
-export([reception/0]).
-export([reception_async/0]).
-export([reception/1]).
-export([disable_rx/0]).

-export([set_frame_timeout/1]).
-export([set_preamble_timeout/1]).
-export([disable_preamble_timeout/0]).

-export([suspend_frame_filtering/0]).
-export([resume_frame_filtering/0]).

-export([read/1]).
-export([write/2]).

-export([rx_ranging_info/0]).
-export([signal_power/0]).
-export([rx_preamble_repetition/0]).
-export([rx_data_rate/0]).
-export([prf_value/0]).
-export([get_conf/0]).
-export([get_rx_metadata/0]).

%%% gen_statem callbacks
-export([init/1]).
-export([callback_mode/0]).
-export([idle/3, rx_on/3, idle_rx/3, idle_to/3]).
-export([terminate/2]).

%--- Records -------------------------------------------------------------------



%--- API -----------------------------------------------------------------------

start_link(_Connector, Params) ->
    gen_statem:start_link({local, ?MODULE}, ?MODULE, Params, []).

start(_Connector, Params) ->
    gen_statem:start({local, ?MODULE}, ?MODULE, Params, []).

stop_link() ->
    gen_statem:stop(?MODULE).

transmit(Frame, Options) ->
    gen_statem:call(?MODULE, {transmit, Frame, Options}).

reception() ->
    case {read(drx_conf), read(rx_fwto)} of
        {#{drx_pretoc := 0}, #{rxfwto := RXFWTO}} ->
            rx_(round(RXFWTO/1000), rxrfto);
        {#{drx_pretoc := PRETOC}, _} ->
            rx_(round(PRETOC/1000), rxpto)
```

```
    end.

reception_async() ->
    case reception() of
        {error, _} = Err ->
            %ct:log("Error? : ~p", [Err]),
            Err;
        Frame ->
            %ct:log("Frame: ~p", [Frame]),
            Metadata = get_rx_metadata(),
            ieee802154_events:rx_event(Frame, Metadata)
    end.

rx_(Timeout, TimeoutError) ->
    case gen_statem:call(?MODULE, {enable_rx, Timeout}, infinity) of
        timeout ->
            {error, TimeoutError};
        affrej ->
            {error, affrej};
        Ret ->
            %ct:log("ret: ~p", [Ret]),
            Ret
    end.

reception(_RxOpts) ->
    reception().

disable_rx() ->
    gen_statem:call(?MODULE, {disable_rx}).

set_frame_timeout(Timeout) when is_float(Timeout) ->
    set_frame_timeout(trunc(Timeout));
set_frame_timeout(Timeout) when is_integer(Timeout) ->
    write(rx_fwto, #{rxfwto => Timeout}),
    write(sys_cfg, #{rxwtoe => 2#1}). % enable receive wait timeout

set_preamble_timeout(Timeout) ->
    write(drx_conf, #{drx_pretoc => Timeout}).

disable_preamble_timeout() ->
    write(drx_conf, #{drx_pretoc => 0}).

suspend_frame_filtering() ->
    write(sys_cfg, #{ffen => 0}).

resume_frame_filtering() ->
    write(sys_cfg, #{ffen => 1}).

read(Reg) ->
    gen_statem:call(?MODULE, {read, Reg}).

write(Reg, Value) ->
    gen_statem:call(?MODULE, {write, Reg, Value}).

%--- API: Getters ----------------------------------------------------------
rx_ranging_info() ->
    #{rng := RNG} = read(rx_finfo),
    RNG.

%% @doc Returns the estimated value of the signal power in dBm
%% cf. user manual section 4.7.2
signal_power() ->
```

```
    C = channel_impulse_resp_pow() , % Channel impulse resonse power value (
        CIR_PWR)
    A = case prf_value() of
            16 -> 113.77;
            64 -> 121.74
        end, % Constant. For PRF of 16 MHz = 113.77, for PRF of 64MHz = 121.74
    N = preamble_acc(), % Preamble accumulation count value (RXPACC but might be
        ajusted)
    Num = C* math:pow(2, 17),
    Dem = math:pow(N, 2),
    Log = math:log10(Num / Dem),
    10 *  Log - A.

preamble_acc() ->
    #{rxpacc := RXPACC} = read(rx_finfo),
    #{rxpacc_nosat := RXPACC_NOSAT} = read(drx_conf),
    if
        RXPACC == RXPACC_NOSAT -> RXPACC;
        true -> RXPACC - 5
    end.

channel_impulse_resp_pow() ->
    #{cir_pwr := CIR_POW} = read(rx_fqual),
    CIR_POW.

%% @doc Gives the value of the PRF in MHz
-spec prf_value() -> 16 | 64.
prf_value() ->
    #{agc_tune1 := AGC_TUNE1} = read(agc_ctrl),
    case AGC_TUNE1 of
        16#8870 -> 16;
        16#889B -> 64
    end.

%% @doc returns the preamble symbols repetition
rx_preamble_repetition() ->
    #{rxpsr := RXPSR} = read(rx_finfo),
    case RXPSR of
        0 -> 16;
        1 -> 64;
        2 -> 1024;
        3 -> 4096
    end.

%% @doc returns the data rate of the received frame in kbps
rx_data_rate() ->
    #{rxbr := RXBR} = read(rx_finfo),
    case RXBR of
        0 -> 110;
        1 -> 850;
        3 -> 6800
    end.

get_conf() ->
    gen_server:call(?MODULE, {get_conf}).

get_rx_metadata() ->
    #{rng := Rng} = read(rx_finfo),
    #{rx_stamp := RxStamp} = read(rx_time),
    #{tx_stamp := TxStamp} = read(tx_time),
    #{rxtofs := Rxtofs} = read(rx_ttcko),
    #{rxttcki := Rxttcki} = read(rx_ttcki),
```

```
    #{snr => snr(),
      prf => prf_value(),
      pre => rx_preamble_repetition(),
      data_rate => rx_data_rate(),
      rng => Rng,
      rx_stamp => RxStamp,
      tx_stamp => TxStamp,
      rxtofs => Rxtofs,
      rxttcki => Rxttcki}.

% Source: https://forum.qorvo.com/t/how-to-calculate-the-signal-to-noise-ratio-snr-of-dw1000/5585/3
snr() ->
    Delta = 87-7.5,
    RSL = signal_power(),
    RSL + Delta.

%--- Internal: gen server callbacks ------------------------------------------

init(#{network := NetworkNode}) ->
    {network_loop, NetworkNode} ! {register, node()},
    ets:new(callback_table, [public, named_table]),
    {ok, idle, #{regs => pmod_uwb_registers:default(),
                 network => NetworkNode,
                 conf => #phy_cfg{}}}.

callback_mode() ->
    [state_functions, state_enter].

idle(enter, _OldState, Data) ->
    {keep_state, Data};
idle({call, From}, {transmit, Frame, Options}, Data) ->
    #{network := NetworkNode, regs := Regs} = Data,
    NewRegs = tx(Frame, Options, NetworkNode, Regs),
    case Options#tx_opts.wait4resp of
        ?ENABLED ->
            {next_state, rx_on, Data#{regs := NewRegs}, {reply, From, ok}};
        ?DISABLED ->
            {keep_state, Data#{regs := NewRegs}, {reply, From, ok}}
    end;
idle({call, From}, {enable_rx, Timeout}, Data) ->
    {next_state, rx_on, Data#{timeout => Timeout, waiting => From}};
idle({call, From}, {disable_rx}, Data) ->
    {keep_state, Data, {reply, From, ok}};
idle(EventType, EventContent, Data) ->
    handle_event(EventType, EventContent, Data).

rx_on(enter, _OldState, #{regs := Regs, timeout := Timeout} = Data) ->
    NewRegs = enable_rx(Regs),
    TimerRef = erlang:start_timer(Timeout, ?MODULE, rx_timeout),
    {keep_state, Data#{regs => NewRegs, timer => TimerRef}};
rx_on({call, From}, {enable_rx, _Timeout}, Data) ->
    {keep_state, Data#{waiting => From}}; % Happens when W4R is enabled
rx_on({call, From}, {disable_rx}, #{regs := Regs, timer := TimerRef} = Data) ->
    erlang:cancel_timer(TimerRef),
    NewRegs = pmod_uwb_registers:update_reg(Regs, sys_ctrl, #{rxenab => ?DISABLED}),
    {next_state, idle, Data#{regs := NewRegs}, {reply, From, ok}};
rx_on(info, {frame, Frame}, #{timer := TimerRef, regs := Regs, network := NetworkNode} = Data) ->
    erlang:cancel_timer(TimerRef),
   %ct:log("Received frame: ~p", [Frame]),
```

```
    NewRegs = handle_rx(Frame, NetworkNode, Regs),
    case Data#{regs := NewRegs} of
        #{waiting := From, regs:= #{sys_status := #{affrej := 0}}} ->
            #{rx_buffer := RawFrame} = pmod_uwb_registers:get_value(NewRegs,
                rx_buffer),
            Reply = {byte_size(RawFrame), RawFrame},
            NewData = maps:remove(waiting, Data),
            {next_state, idle, NewData#{regs => NewRegs}, {reply, From, Reply}};
        #{waiting := From, regs:= #{sys_status := #{affrej := 1}}} ->
            NewData = maps:remove(waiting, Data),
            {next_state, idle, NewData#{regs => NewRegs}, {reply, From, affrej}};
        _ ->
            {next_state, idle_rx, Data#{regs => NewRegs}}
    end;
rx_on(info, {timeout, _, rx_timeout}, #{regs := Regs} = Data) ->
    NewRegs = handle_timeout(Regs),
    case Data of
        #{waiting := From} ->
            NewData = maps:remove(waiting, Data),
            {next_state, idle, NewData#{regs => NewRegs}, {reply, From, timeout}};
        _ ->
            {next_state, idle_to, Data#{regs => NewRegs}}
    end;
rx_on(EventType, EventContent, Data) ->
    handle_event(EventType, EventContent, Data).

idle_rx(enter, _OldState, Data) ->
    {keep_state, Data};
idle_rx({call, From}, {enable_rx, _Timeout}, #{regs := Regs} = Data) ->
    case pmod_uwb_registers:get_value(Regs, sys_status) of
        #{affrej := 1} ->
            {next_state, idle, Data, {reply, From, affrej}};
        _ ->
            #{rx_buffer := RawFrame} = pmod_uwb_registers:get_value(Regs,
                rx_buffer),
            Reply = {byte_size(RawFrame), RawFrame},
            NewData = maps:remove(waiting, Data),
            {next_state, idle, NewData#{regs => Regs}, {reply, From, Reply}}
    end;
idle_rx(EventType, EventContent, Data) ->
    handle_event(EventType, EventContent, Data).

idle_to(enter, _OldState, Data) ->
    {keep_state, Data};
idle_to({call, From}, {enable_rx, _Timeout}, Data) ->
    {next_state, idle, Data, {reply, From, timeout}}.

handle_event({call, From}, {read, Reg}, #{regs := Regs} = Data) ->
    Val = pmod_uwb_registers:get_value(Regs, Reg),
    {keep_state, Data, {reply, From, Val}};
handle_event({call, From}, {write, Reg, Value}, #{regs := Regs} = Data) ->
    NewRegs = pmod_uwb_registers:update_reg(Regs, Reg, Value),
    {keep_state, Data#{regs => NewRegs}, {reply, From, ok}};
handle_event({call, From}, {get_conf}, #{conf := Conf} = Data) ->
    {keep_state, Data, {reply, From, Conf}};
handle_event(info, Event, Data) ->
    %ct:log("Event skipped: ~p", [Event]),
    {keep_state, Data};
handle_event(EventType, EventContent, _Data) ->
    error({unknown_event, EventType, EventContent}).

terminate(Reason, _) ->
```

```
    io:format("Terminate: ~w", [Reason]).

%--- Internal ---------------------------------------------------------------

tx(Frame, Options, NetworkNode, Regs) ->
    Rng = Options#tx_opts.ranging,
    PhyFrame = {Rng, Frame},
    %ct:log("Tx frame ~p", [Frame]),
    {network_loop, NetworkNode} ! {tx, node(), PhyFrame},
    pmod_uwb_registers:update_reg(Regs, tx_fctrl, #{tr => Rng}).

enable_rx(Regs) ->
    NewRegs = pmod_uwb_registers:update_reg(Regs, sys_ctrl, #{rxenab => 1}),
    pmod_uwb_registers:update_reg(NewRegs, sys_status, #{rxfcg => 0, affrej => 0}).

handle_rx({_, <<_:5/bitstring, ?FTYPE_ACK:3, _/bitstring>>=RawFrame}, _, Regs) ->
    %ct:log("Received Ack"),
    NewRegs1 = pmod_uwb_registers:update_reg(Regs, sys_cfg, #{rxenab => ?DISABLED}),
    pmod_uwb_registers:update_reg(NewRegs1, rx_buffer, #{rx_buffer => RawFrame});
handle_rx(Frame, NetworkNode, #{sys_cfg := #{ffen := ?ENABLED}} = Regs) ->
    {Rng, RawFrame} = Frame,
    #{short_addr := ShortAddress} = pmod_uwb_registers:get_value(Regs, panadr),
    #{eui := ExtAddress} = pmod_uwb_registers:get_value(Regs, eui),
    case check_address(RawFrame, ShortAddress, ExtAddress) of
        ok ->
            AckRegs = ack_reply(RawFrame, NetworkNode, Regs),
            NewRegs = pmod_uwb_registers:update_reg(AckRegs, rx_finfo, #{rng => Rng}),
            NewRegs1 = pmod_uwb_registers:update_reg(NewRegs, sys_cfg, #{rxenab => ?DISABLED}),
            pmod_uwb_registers:update_reg(NewRegs1, rx_buffer, #{rx_buffer => RawFrame});
        _ ->
            NewRegs = pmod_uwb_registers:update_reg(Regs, sys_cfg, #{rxenab => ?DISABLED}),
            pmod_uwb_registers:update_reg(NewRegs, sys_status, #{affrej => 1})
    end;
handle_rx(Frame, _, Regs) ->
    {Rng, RawFrame} = Frame,
    NewRegs = pmod_uwb_registers:update_reg(Regs, rx_finfo, #{rng => Rng}),
    NewRegs1 = pmod_uwb_registers:update_reg(NewRegs, sys_cfg, #{rxenab => ?DISABLED}),
    pmod_uwb_registers:update_reg(NewRegs1, rx_buffer, #{rx_buffer => RawFrame}).

handle_timeout(Regs) ->
    pmod_uwb_registers:update_reg(Regs, sys_status, #{rxenab => 0}).

check_address(Frame, ShortAddress, ExtAddress) -> % This will need to check the PAN and accept broadcast address at some point
    {_, MacHeader, _} = mac_frame:decode(Frame),
    case MacHeader#mac_header.dest_addr of
        ShortAddress -> ok;
        ExtAddress -> ok;
        _ -> continue
    end.

ack_reply(_, _, #{sys_cfg := #{autoack := 0}} = Regs) ->
    Regs;
ack_reply(Frame, NetworkNode, Regs) ->
    <<_:2, _ACKREQ:1, _/bitstring>> = Frame,
```

```
    % io:format("Ack req: ~w ~n ~w", [ACKREQ, Frame]),
    case Frame of
        <<?FTYPE_ACK:2, _/bitstring>> ->
            ok;
        <<_:2, ?ENABLED:1, _:13, Seqnum:8, _/bitstring>> ->
            % io:format("Ack requested~n"),
            Ack = mac_frame:encode_ack(?DISABLED, Seqnum),
            %io:format("Ack reply from ieee~n"),
            tx(Ack, #tx_opts{}, NetworkNode, Regs);
        _ ->
            % io:format("No Ack requested~n"),
            Regs
    end.
```

```
-module(mock_phy).
-behaviour(gen_server).

-export([start_link/2]).
-export([start/2]).
-export([stop_link/0]).

-export([transmit/2]).
-export([reception_async/0]).
-export([reception/0]).
-export([reception/1]).
-export([disable_rx/0]).

-export([set_frame_timeout/1]).
-export([set_preamble_timeout/1]).
-export([disable_preamble_timeout/0]).

-export([suspend_frame_filtering/0]).
-export([resume_frame_filtering/0]).

-export([read/1]).
-export([write/2]).

-export([rx_ranging_info/0]).
-export([signal_power/0]).
-export([rx_preamble_repetition/0]).
-export([rx_data_rate/0]).
-export([prf_value/0]).
-export([get_conf/0]).

%%% gen_server callbacks
-export([init/1]).
-export([handle_call/3]).
-export([handle_cast/2]).

%--- Include ---------------------------------------------------------

-include("../src/pmod_uwb.hrl").

%--- Macros ---------------------------------------------------------

-define(NAME, mock_phy).

% --- API -----------------------------------------

start_link(_Connector, Params) ->
    gen_server:start_link({local, ?NAME}, ?MODULE, Params, []).
```

```
start(_Connector, Params) ->
    gen_server:start({local, ?NAME}, ?MODULE, Params, []).

stop_link() ->
    gen_server:stop(?NAME).

transmit(Data, Options) ->
    gen_server:call(?NAME, {transmit, Data, Options}).

reception_async() ->
    Frame =  gen_server:call(?NAME, {reception}),
    Metadata = #{snr => 10.0,
                 prf => 4,
                 pre => 16,
                 data_rate => 1,
                 rng => ?DISABLED,
                 rx_stamp => 1,
                 tx_stamp => 1,
                 rxtofs => 1,
                 rxttcki => 1},
    ieee802154_events:rx_event(Frame, Metadata).

reception() ->
    gen_server:call(?NAME, {reception}).

reception(_) ->
    gen_server:call(?NAME, {reception}).

set_frame_timeout(Timeout) ->
    write(rx_fwto, #{rxfwto => Timeout}),
    write(sys_cfg, #{rxwtoe => 2#1}). % enable receive wait timeout

set_preamble_timeout(Timeout) ->
    write(drx_conf, #{drx_pretoc => Timeout}).

disable_preamble_timeout() ->
    write(drx_conf, #{drx_pretoc => 0}).

suspend_frame_filtering() ->
    write(sys_cfg, #{ffen => 0}).

resume_frame_filtering() ->
    write(sys_cfg, #{ffen => 1}).

read(Reg) ->
    gen_server:call(?NAME, {read, Reg}).

write(Reg, Val) ->
    gen_server:call(?NAME, {write, Reg, Val}).

disable_rx() ->
    gen_server:call(?NAME, {rx_off}).

%--- API: Getters -------------------------------------------------------------
rx_ranging_info() ->
    #{rng := RNG} = read(rx_finfo),
    RNG.

%% @doc Returns the estimated value of the signal power in dBm
%% cf. user manual section 4.7.2
signal_power() ->
```

```
    C = channel_impulse_resp_pow() , % Channel impulse resonse power value (
        CIR_PWR)
    A = case prf_value() of
            16 -> 113.77;
            64 -> 121.74
        end, % Constant. For PRF of 16 MHz = 113.77, for PRF of 64MHz = 121.74
    N = preamble_acc(), % Preamble accumulation count value (RXPACC but might be
        ajusted)
    Num = C* math:pow(2, 17),
    Dem = math:pow(N, 2),
    Log = math:log10(Num / Dem),
    10 *  Log - A.

preamble_acc() ->
    #{rxpacc := RXPACC} = read(rx_finfo),
    #{rxpacc_nosat := RXPACC_NOSAT} = read(drx_conf),
    if
        RXPACC == RXPACC_NOSAT -> RXPACC;
        true -> RXPACC - 5
    end.

channel_impulse_resp_pow() ->
    #{cir_pwr := CIR_POW} = read(rx_fqual),
    CIR_POW.

%% @doc Gives the value of the PRF in MHz
-spec prf_value() -> 16 | 64.
prf_value() ->
    #{agc_tune1 := AGC_TUNE1} = read(agc_ctrl),
    case AGC_TUNE1 of
        16#8870 -> 16;
        16#889B -> 64
    end.

%% @doc returns the preamble symbols repetition
rx_preamble_repetition() ->
    #{rxpsr := RXPSR} = read(rx_finfo),
    case RXPSR of
        0 -> 16;
        1 -> 64;
        2 -> 1024;
        3 -> 4096
    end.

%% @doc returns the data rate of the received frame in kbps
rx_data_rate() ->
    #{rxbr := RXBR} = read(rx_finfo),
    case RXBR of
        0 -> 110;
        1 -> 850;
        3 -> 6800
    end.

get_conf() ->
    gen_server:call(?NAME, {get_conf}).

%--- gen_server callbacks ------------------------------------------------------
init(_Params) ->
    {ok, #{regs => pmod_uwb_registers:default(),
           conf => #phy_cfg{}}}.

handle_call({transmit, Data, Options}, _From, State) -> {reply, tx(Data, Options),
```

```
     State};
handle_call({reception}, _From, State) -> {reply, rx(), State};
handle_call({read, Reg}, _From, #{regs := Regs} = State) -> {reply, maps:get(Reg,
    Regs), State};
handle_call({write, Reg, Val}, _From, #{regs := Regs} = State) -> {reply, ok,
    State#{regs => pmod_uwb_registers:update_reg(Regs, Reg, Val)}};
handle_call({rx_off}, _From, State) -> {reply, ok, State};
handle_call({get_conf}, _From, #{conf := Conf} = State) -> {reply, Conf, State};
handle_call(_Request, _From, _State) -> error(not_implemented).

handle_cast(_Request, _State) ->
  error(not_implemented).


% --- Internal -----------------------------------------
tx(_Data, _Options) ->
    ok.

rx() ->
    {14, <<16#6188:16, 0:8, 16#CADE:16, "XR", "XT", "Hello">>}.
```

```
-module(mock_top_layer).

-behaviour(gen_server).

-include("../src/mac_frame.hrl").

-include_lib("common_test/include/ct.hrl").

%%% EXPORT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% API functions
-export([start/0]).
-export([rx_frame/4]).
-export([dump/0]).
-export([stop/0]).

%%% gen_server callbacks
-export([init/1]).
-export([handle_call/3]).
-export([handle_cast/2]).
-export([handle_info/2]).
-export([terminate/2]).

%%% MACROS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

-define(RCVR_ADDR, <<16#CAFEDECA00000003:64>>).
-define(MDL_ADDR, <<16#CAFEDECA00000002:64>>).

%%% API %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
start() ->
    gen_server:start({local, ?MODULE}, ?MODULE, [], []).

rx_frame(Frame, _, _, _) ->
    gen_server:cast(?MODULE, {rx, Frame}).

dump() ->
    gen_server:call(?MODULE, {dump}).

stop() ->
    gen_server:stop(?MODULE).

```

```
%%% gen_server callbacks %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
init(_) ->
    {ok, #{received => #{}}}.

handle_call({dump}, _From, #{received := Received} = State) ->
    Ret = lists:sort(fun({_, MH1, _}, {_, MH2, _}) ->
                         MH1#mac_header.seqnum < MH2#mac_header.seqnum
                     end, maps:values(Received)),
    {reply, {ok, Ret}, State#{received => []}};
handle_call(_, _, _) ->
  error(not_implemented).

handle_cast({rx, {FC, MH, Payload}=Frame}, #{received := Received} = State)
  when FC#frame_control.frame_type == ?FTYPE_DATA ->
    Seqnum = MH#mac_header.seqnum,
    case Received of
        #{Seqnum := _} ->
            {noreply, State};
        _ ->
            case MH#mac_header.dest_addr of
                ?MDL_ADDR ->
                    NewMH = MH#mac_header{seqnum = Seqnum + 10,
                                          src_addr = ?MDL_ADDR,
                                          dest_addr = ?RCVR_ADDR},
                    NewFrame = {FC, NewMH, Payload},
                    ieee802154:transmission(NewFrame);
                _ ->
                    ok
            end,
            {noreply, State#{received => maps:put(Seqnum, Frame, Received)}}
    end;
handle_cast({rx, _}, State) ->
    {noreply, State};
handle_cast(_, _) ->
  error(not_implemented).

handle_info(_, _) ->
  error(not_implemented).

terminate(_, _) ->
    ok.
```

```
-module(network_simulation).

-behaviour(application).

-include_lib("common_test/include/ct.hrl").

-include("../src/mac_frame.hrl").

%%% EXPORTS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% application callbacks
-export([start/2]).
-export([stop/1]).

%%% application callbacks %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
-spec start(_, Args) -> {ok, pid()} when Args :: #{loss => boolean()}.
start(_, Args) ->
    Loss = maps:get(loss, Args, false),
    StartData =
        #{
```

```
            nodes => sets:new(),
            exchanges => #{},
            loss => Loss
        },
    LoopPid = spawn(fun() -> loop(ready, StartData) end),
    register(network_loop, LoopPid),
    {ok, LoopPid}.

stop(_) ->
    network_loop ! {stop},
    unregister(network_loop).

%%%% Internal %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

-spec loop(State, Data) -> ok when
    State :: ready | blocked,
    Data :: #{nodes := NodeMap, loss := boolean()},
    NodeMap :: #{node() => sets:set()}.
loop(ready, #{nodes := Nodes} = Data) ->
    receive
        {ping, Pid, Node} ->
            {Pid, Node} ! pong,
            loop(ready, Data);
        {register, Name} ->
            NewNodes = sets:add_element(Name, Nodes),
            loop(ready, Data#{nodes => NewNodes});
        {tx, From, Frame} ->
            do_broadcast(Data, From, Frame);
        {stop} ->
            ok
    end;
loop(blocked, #{nodes := Nodes} = Data) ->
    receive
        {register, _Name} ->
            loop(ready, Data);
        % Once a frame has been blocked => pass through
        {tx, From, Frame} ->
            broadcast(sets:to_list(Nodes), From, Frame),
            loop(blocked, Data);
        {stop} ->
            ok;
        OtherEvent ->
            error(wrong_event_in_blocked, OtherEvent)
    end.

do_broadcast(#{loss := false, nodes := Nodes} = Data, From, Frame) ->
    broadcast(sets:to_list(Nodes), From, Frame),
    loop(ready, Data);
do_broadcast(#{loss := true} = Data, From, Frame) ->
    {_, RawFrame} = Frame,
    #{nodes := Nodes, exchanges := Exchanges} = Data,
    {_, MH, _} = mac_frame:decode(RawFrame),
    #mac_header{dest_addr = DestAddr} = MH,
    Key = {From, DestAddr},
    case maps:get(Key, Exchanges, {0, 0}) of
        {MemorySeen, MemorySeen} ->
            NewExch = maps:put(Key, {0, MemorySeen + 1}, Exchanges),
            loop(blocked, Data#{exchanges => NewExch});
        {RndSeen, MemorySeen} ->
            broadcast(sets:to_list(Nodes), From, Frame),
            NewExch = maps:put(Key, {RndSeen + 1, MemorySeen}, Exchanges),
            loop(ready, Data#{exchanges => NewExch})
```

```
    end.

broadcast([], _, _) ->
    ok;
broadcast([From | T], From, Frame) ->
    broadcast(T, From, Frame);
broadcast([Node | T], From, Frame) ->
    {mock_phy_network, Node} ! {frame, Frame},
    broadcast(T, From, Frame).
```

```
-module(pmod_uwb_registers).

-export([default/0]).
-export([update_reg/3]).
-export([get_value/2]).

-spec default() -> map().
default() ->
    #{eui => #{eui => <<16#FFFFFFFF00000000:64>>}, % 0x01
      panadr => #{pan_id => <<16#FFFFFFFF:16>>, short_addr => <<16#FFFFFFFF:16>>},
           % 0x03
      sys_cfg => #{aackpend => 0, % 0x04
                   autoack => 0,
                   rxautr => 0,
                   rxwtoe => 0,
                   rxm110k => 0,
                   dis_stxp => 0,
                   phr_mode => 0,
                   fcs_init2f => 0,
                   dis_rsde => 0,
                   dis_phe => 0,
                   dis_drxb => 1,
                   dis_fce => 0,
                   spi_edge => 0,
                   hirq_pol => 1,
                   ffa5 => 0,
                   ffa4 => 0,
                   ffar => 0,
                   ffam => 0,
                   ffaa => 0,
                   ffad => 0,
                   ffab => 0,
                   ffbc => 0,
                   ffen => 0},
      tx_fctrl => #{txboffs => 0, % 0x08
                    pe => 1,
                    txpsr => 1,
                    txprf => 1,
                    tr => 0,
                    txbr => 2,
                    r => 0,
                    tfle => 0,
                    tflen => 12},
      rx_fwto => #{rxfwto => 0}, % 0x0C
      sys_ctrl => #{sfcst => 0, % 0x0D
                    txstrt => 0,
                    txdlys => 0,
                    trxoff => 0,
                    wait4resp => 0,
                    rxenab => 0,
                    rxdlye => 0,
```

```
                    hrbpt => 0},
      sys_status => #{irqs => 0, % 0x0F
                      cplock => 0,
                      esyncr => 0,
                      aat => 0,
                      txfrb => 0,
                      txprs => 0,
                      pxphs => 0,
                      txfrs => 0,
                      rxprd => 0,
                      rxsfdd => 0,
                      ldedone => 0,
                      rxphd => 0,
                      rxphe => 0,
                      rxdfr => 0,
                      rxfcg => 0,
                      rxfce => 0,
                      rxrfsl => 0,
                      rxrfto => 0,
                      ldeerr => 0,
                      rxovrr => 0,
                      rxpto => 0,
                      gpioirq => 0,
                      slp2init => 0,
                      rfpll_ll => 0,
                      clkpll_ll => 0,
                      rxsfdto => 0,
                      hpdwarn => 0,
                      txberr => 0,
                      affrej => 0,
                      hsrbp => 0,
                      icrbp => 0,
                      rxrscs => 0,
                      rxprej => 0,
                      txpute => 0},
      rx_finfo => #{rxpacc => 1025, % 0x10
                     rxpsr => 0,
                     rxprfr => 0,
                     rng => 0,
                     rxbr => 0,
                     rxfle => 0,
                     rxflen => 0},
      rx_buffer => #{rx_buffer => <<>>}, % 0x11
      rx_fqual => #{fp_ampl2 => 0, % 0x12
                    std_noise => 0,
                    cir_pwr => 1,
                    pp_ampl3 => 0},
      rx_ttcki => #{rxttcki => 0}, % 0x13
      rx_ttcko => #{rmspdel => 0, % 0x14
                    rxtofs => 0,
                    rcphase => 0},
      rx_time => #{rx_stamp => 0, % 0x15
                   fp_index => 0,
                   fp_ampl1 => 0,
                   rx_rawst => 0},
      tx_time => #{tx_stamp => 0, % 0x17
                   tx_rawst => 0},
      rx_sniff => #{sniff_offt => 0, sniff_ont => 0}, % 0x1D
      agc_ctrl => #{agc_ctrl1 => #{dis_am => 1},
                    agc_tune1 => 16#8870,
                    agc_tune2 => 16#2502A907,
                    agc_tune3 => 16#0035,
```

```
                    agc_stat1 => #{edv2 => 0,
                                   edg1 => 1}},
      % DRX_CONF isn't complete yet
      drx_conf => #{drx_pretoc => 0, % 0x27
                    rxpacc_nosat => 0},
      % PMSC isn't complete yet
      pmsc => #{pmsc_ctrl0 => #{},  % 0x36
                pmsc_ctrl1 => #{arx2init => 0}}
     }.

-spec update_reg(Regs::map(), Reg::atom(), NewVal::atom()|map()) -> map().
update_reg(Regs, Reg, NewVal) ->
    OldVal = maps:get(Reg, Regs),
    if is_map(OldVal) -> maps:put(Reg, maps:merge(OldVal, NewVal), Regs);
       true -> maps:put(Reg, NewVal, Regs) end.

-spec get_value(Regs::map(), Reg::atom()) -> atom()|map().
get_value(Regs, Reg) ->
    maps:get(Reg, Regs).
```

A.13 Mac layer code

```
-module(ieee802154).
-behaviour(gen_server).

%%% @headerfile "ieee802154.hrl"

% API
-export([start_link/1]).
-export([start/1]).
-export([stop_link/0]).
-export([stop/0]).

-export([transmission/1]).
-export([transmission/2]).
-export([reception/0]).

-export([rx_on/0]).
-export([rx_off/0]).

-export([get_pib_attribute/1]).
-export([set_pib_attribute/2]).

-export([reset/1]).

% gen_server callbacks
-export([init/1]).
-export([terminate/2]).
-export([code_change/4]).
-export([handle_call/3]).
-export([handle_cast/2]).


% Includes
-include("ieee802154.hrl").
-include("ieee802154_pib.hrl").
-include("mac_frame.hrl").
```

```erlang

%--- Types -------------------------------------------------------------------

-type state() :: #{phy_layer := module(),
                   duty_cycle := gen_duty_cycle:state(),
                   pib := pib_state(),
                   _:=_}.

%--- API ---------------------------------------------------------------------

%% @doc Starts the IEEE 812.15.4 stack and creates a link
%%
%% ```
%% The following code will start the stack with the default parameters
%% 1> ieee802154:start_link(#ieee_parameters{}).
%%
%% Using a custom callback function
%% 2> ieee802154:start_link(#ieee_parameters{input_callback = fun callback/4}).
%%
%% Using a custom phy module
%% 3> ieee802154:start_link(#ieee_parameters{phy_layer = mock_phy_network}).
%% '''
%%
%% @param Params: A map containing the parameters of the IEEE stack
%%
%% @end
-spec start_link(Params) -> {ok, pid()} | {error, any()} when
      Params :: ieee_parameters().
start_link(Params) ->
    gen_server:start_link({local, ?MODULE}, ?MODULE, Params, []).

%% @doc Same as start_link/1 but no link is created
%% @end
-spec start(Params) -> {ok, pid()} | {error, any()} when
      Params :: ieee_parameters().
start(Params) ->
    gen_server:start({local, ?MODULE}, ?MODULE, Params, []).

stop_link() ->
    gen_server:stop(?MODULE).

stop() -> gen_server:stop(?MODULE).

%% @doc
%% @equiv transmission(Frame, 0)
%% @end
-spec transmission(Frame) -> Result when
      Frame        :: frame(),
      Result       :: {ok, Ranging} | {error, Error},
      Ranging      :: ranging_informations(),
      Error        :: tx_error().
transmission(Frame) ->
    transmission(Frame, ?NON_RANGING).

%% @doc Performs a transmission on the defined IEEE 802.15.4 stack
%% When ranging has been activated for the frame, the second element of the
%% tuple contains different values that can be used for ranging operations
%% For more informations please consult the IEEE 802.15.4 standard.
%% Note that the variable `Timestamp' is omited because its value is the same
%% as `Ranging counter start'
%%
%% When Ranging isn't activated, the 2nd element of the tuple shall be ignored
```

```erlang
%% ```
%% Ranging not activated for transmission
%% 1> ieee802154:transmission(Frame, ?NON_RANGING).
%%
%% Activate ranging for the transmission
%% 2> ieee802154:transmission(Frame, ?ALL_RANGING).
%% '''
%% @end
-spec transmission(Frame, Ranging) -> Result when
      Frame        :: frame(),
      Ranging      :: ranging_tx(),
      Result       :: {ok, RangingInfos} | {error, Error},
      RangingInfos :: ranging_informations(),
      Error        :: tx_error().
transmission(Frame, Ranging) ->
    {FH, _, _} = Frame,
    case FH of
        #frame_control{dest_addr_mode = ?NONE, src_addr_mode = ?NONE} ->
            {error, invalid_address};
        _ ->
            gen_server:call(?MODULE,
                            {tx, Frame, Ranging},
                            infinity)
    end.

%% @doc Performs a reception on the IEEE 802.15.4 stack
%% @deprecated This function will be deprecated
%% @end
-spec reception() -> Result when
      Result :: {ok, frame()} | {error, atom()}.
reception() ->
    gen_server:call(?MODULE, {rx}, infinity).

%% @doc Turns on the continuous reception
%% Ranging is by default switched on
%% @end
-spec rx_on() -> Result when
      Result :: ok | {error, atom()}.
rx_on() ->
    gen_server:call(?MODULE, {rx_on}).

%% @doc Turns off the continuous reception
%% @end
rx_off() ->
    gen_server:call(?MODULE, {rx_off}).

%% @doc Get the value of a PIB attribute
%% @end
-spec get_pib_attribute(Attribute) -> Value when
      Attribute :: pib_attribute(),
      Value     :: term().
get_pib_attribute(Attribute) ->
    gen_server:call(?MODULE, {get, Attribute}).


%% @doc Set the value of a PIB attribute
%% @end
-spec set_pib_attribute(Attribute, Value) -> ok when
      Attribute :: pib_attribute(),
      Value     :: term().
set_pib_attribute(Attribute, Value) ->
    gen_server:call(?MODULE, {set, Attribute, Value}).
```

```

-spec reset(SetDefaultPIB) -> Result when
      SetDefaultPIB :: boolean(),
      Result :: ok.
reset(SetDefaultPIB) ->
    gen_server:call(?MODULE, {reset, SetDefaultPIB}).

%--- gen_statem callbacks -----------------------------------------------

-spec init(Params) -> {ok, State} when
      Params :: ieee_parameters(),
      State  :: state().
init(Params) ->
    {ok, _GenEvent} = gen_event:start_link({local, ?GEN_EVENT}),
    PhyMod = Params#ieee_parameters.phy_layer,
    InputCallback = Params#ieee_parameters.input_callback,
    write_default_conf(PhyMod),

    ok = ieee802154_events:start(#{input_callback => InputCallback}),

    DutyCycleState = gen_duty_cycle:start(Params#ieee_parameters.duty_cycle,
                                          PhyMod),

    Data = #{phy_layer => PhyMod,
             duty_cycle => DutyCycleState,
             pib => ieee802154_pib:init(PhyMod),
             ranging => ?DISABLED},
    {ok, Data}.

-spec terminate(Reason, State) -> ok when
      Reason :: term(),
      State :: state().
terminate(Reason, #{duty_cycle := GenDutyCycleState}) ->
    ieee802154_events:stop(),
    gen_event:stop(?GEN_EVENT),
    gen_duty_cycle:stop(GenDutyCycleState, Reason).

code_change(_, _, _, _) ->
    error(not_implemented).

-spec handle_call(_, _, State) -> Result when
      State   :: state(),
      Result  :: {reply, term(), State}.
handle_call({rx_on}, _From, State) ->
    #{duty_cycle := DCState} = State,
    case gen_duty_cycle:turn_on(DCState) of
        {ok, NewDutyCycleState} ->
            {reply, ok, State#{duty_cycle => NewDutyCycleState}};
        {error, NewDutyCycleState, Error} ->
            {reply, {error, Error}, State#{duty_cycle => NewDutyCycleState}}
    end;
handle_call({rx_off}, _From, #{duty_cycle := DCState} = State) ->
    NewDCState = gen_duty_cycle:turn_off(DCState),
    {reply, ok, State#{duty_cycle => NewDCState}};
handle_call({tx, Frame, Ranging}, _From, State) ->
    #{duty_cycle := DCState, pib := Pib} = State,
    {FrameControl, MacHeader, Payload} = Frame,
    EncFrame = mac_frame:encode(FrameControl, MacHeader, Payload),
    case gen_duty_cycle:tx_request(DCState, EncFrame, Pib, Ranging) of
        {ok, NewDCState, RangingInfos} ->
            timer:sleep(100), % FIXME: IFS
            {reply, {ok, RangingInfos}, State#{duty_cycle => NewDCState}};
```

```
        {error, NewDCState, Error} ->
            {reply, {error, Error}, State#{duty_cycle => NewDCState}}
        end;
handle_call({get, Attribute}, _From, State) ->
    #{pib := Pib} = State,
    case ieee802154_pib:get(Pib, Attribute) of
        {error, Error} ->
            {reply, {error, Error}, State};
        Value ->
            {reply, Value, State}
    end;
handle_call({set, Attribute, Value}, _From, State) ->
    #{pib := Pib} = State,
    case ieee802154_pib:set(Pib, Attribute, Value) of
        {ok, NewPib} ->
            {reply, ok, State#{pib => NewPib}};
        {error, NewPib, Error} ->
            {reply, {error, Error}, State#{pib => NewPib}}
    end;
handle_call({reset, SetDefaultPIB}, _From, State) ->
    #{phy_layer := PhyMod, pib := Pib, duty_cycle := DCState} = State,
    NewState = case SetDefaultPIB of
                   true ->
                       PhyMod:write(panadr, #{pan_id => <<16#FFFF:16>>,
                                              short_addr => <<16#FFFF:16>>}),
                       State#{pib => ieee802154_pib:reset(Pib)};
                   _ ->
                       State
               end,
    NewDCState = gen_duty_cycle:turn_off(DCState),
    {reply, ok, NewState#{duty_cycle => NewDCState, ranging => ?DISABLED}};
handle_call(_Request, _From, _State) ->
    error(call_not_recognized).

handle_cast(_, _) ->
    error(not_implemented).

%--- Internal -----------------------------------------------------------------
-spec write_default_conf(PhyMod :: module()) -> ok.
write_default_conf(PhyMod) ->
    PhyMod:write(rx_fwto, #{rxfwto => ?MACACKWAITDURATION}),
    PhyMod:write(sys_cfg, #{ffab => 1,
                            ffad => 1,
                            ffaa => 1,
                            ffam => 1,
                            ffen => 1,
                            autoack => 1,
                            rxwtoe => 1}).
```

```
%--- Macros -------------------------------------------------------------------

%--- MCPS-DATA.indication Parameters

% Ranging Received values:
-define(NO_RANGING_REQUESTED, 0).
-define(RANGING_REQUESTED_BUT_NOT_SUPPORTED, 1).
-define(RANGING_ACTIVE, 2).

% Ranging Transmission values:
-define(NON_RANGING, 0).
-define(ALL_RANGING, 1).
```

```
% PHY_HEADER_ONLY => Not supported in our case

% CSMA constants
-define(MACMAXFRAMERETRIES, 5).
-define(MACACKWAITDURATION, 4000).  % works with 2000  s  but calculations give me 4081 s
% -define(MACACKWAITDURATION, 2000).  % works with 2000  s  but calculations give me 4081 s
%
-define(GEN_EVENT, gen_event).

%--- Types -------------------------------------------------------------
%--- Record types
-record(ieee_parameters, {duty_cycle = duty_cycle_non_beacon :: module(),
                          phy_layer = pmod_uwb :: module(),
                          input_callback = fun(_, _, _, _) -> ok end :: input_callback()}).


-record(ranging_informations, {ranging_received = ?NO_RANGING_REQUESTED :: ranging_received() | boolean(),
                               ranging_counter_start = 0                :: integer(),
                               ranging_counter_stop = 0                 :: integer(),
                               ranging_tracking_interval = 0            :: integer(),
                               ranging_offset = 0                       :: integer(),
                               ranging_FOM = <<16#00:8>>                :: bitstring()}).

-type ranging_informations() :: #ranging_informations{}.

% For now security isn't enabled
-record(security, {security_level = 0       :: integer(),
                   key_id_mode = 0          :: integer(),
                   key_source = <<16#00:8>> :: bitstring()}).

-type security() :: #security{}.

%--- IEEE 802.15.4 parameter types
-export_type([ieee_parameters/0, ranging_informations/0, security/0, input_callback/0, ranging_tx/0, tx_error/0]).

-type ranging_received() :: ?NO_RANGING_REQUESTED | ?RANGING_REQUESTED_BUT_NOT_SUPPORTED | ?RANGING_ACTIVE.
-type ranging_tx() :: ?NON_RANGING | ?ALL_RANGING. % PHY_HEADER_ONLY no used in our case

% *** indicates unusefull parameters for higher layers for now
-type input_callback() :: fun((Frame                     :: mac_frame:frame(),
                               LQI                       :: integer(),
                               % UWBPRF                    :: gen_mac_layer:uwb_PRF(),
                               Security                  :: security(),
                               % UWBPreambleRepetitions :: pmod_uwb:uwb_preamble_symbol_repetition(),
                               % DataRate                  :: pmod_uwb:data_rate(),
                               Ranging                   :: ranging_informations())
                              -> ok).

```

```
-type ieee_parameters() :: #ieee_parameters{}.

-type tx_error() :: invalid_address | invalid_gts | transaction_overflow |
    transaction_expired | no_ack | frame_too_long | channel_access_failure.
```

```
-module(ieee802154_events).

-behaviour(gen_event).

%--- Exports ------------------------------------------------------------------

-export([start/1]).
-export([stop/0]).
-export([rx_event/2]).

% gen_event callbacks
-export([init/1]).
-export([handle_event/2]).
-export([handle_call/2]).
-export([handle_info/2]).
-export([terminate/2]).

%--- Includes -----------------------------------------------------------------

-include("ieee802154.hrl").
-include("pmod_uwb.hrl").

%--- Records ------------------------------------------------------------------

-record(state, {input_callback :: ieee802154:input_callback()}).

%--- Types --------------------------------------------------------------------

-type state() :: #state{}.

%--- API ----------------------------------------------------------------------
% TODO:
% * Notify an event
% * Subscribe to an event => Wait for an event to happen
% * Add a callback for an event ? => Are these events ?
-spec start(Args :: map()) -> ok.
start(Args) ->
    gen_event:add_handler(?GEN_EVENT, ?MODULE, Args).

stop() ->
    gen_event:delete_handler(?GEN_EVENT, ?MODULE, []).

% @doc triggers a rx event
% -spec rx_event(Frame, Metadata) -> ok when
%       Frame :: {integer(), bitstring()},
%       Metadata :: #{snr := float(),
%                     prf := uwb_PRF(),
%                     pre := uwb_preamble_symbol_repetition(),
%                     data_rate := data_rate(),
%                     rng := flag(),
%                     rx_stamp := integer(),
%                     tx_stamp := integer(),
%                     rxtofs := integer(),
%                     rxttcki := integer()}.
rx_event({_, Frame}, Metadata) ->
    gen_event:notify(?GEN_EVENT, {rx, Frame, Metadata}).
```

```erlang

%--- gen_event callbacks ------------------------------------------------------
-spec init(InitArgs :: map()) ->
    {ok, State :: state()}.
init(State) ->
    #{input_callback := InputCallback} = State,
    {ok, #state{input_callback = InputCallback}}.

handle_event({rx, Frame, Metadata}, State) ->
    #state{input_callback = InputCallback} = State,
    DecodedFrame = mac_frame:decode(Frame),
    #{snr := Snr,
      rng := Rng,
      rx_stamp := RxStamp,
      tx_stamp := TxStamp,
      rxtofs := Rxtofs,
      rxttcki := Rxttcki} = Metadata,
    RngInfo = rng_infos(Rng, RxStamp, TxStamp,Rxtofs, Rxttcki),
    InputCallback(DecodedFrame, Snr, #security{}, RngInfo),
    {ok, State};
handle_event(_Event, State) ->
    {ok, State}.

handle_call(_, State) ->
    % TODO user should be able to register a callback
    {ok, ok, State}.

handle_info(_, State) ->
    % TODO: nothing here
    {ok, State}.

terminate(_, _) ->
    ok.

%--- Internal -----------------------------------------------------------------

rng_infos(?ENABLED, RxStamp, TxStamp, Rxtofs, Rxttcki) ->
    #ranging_informations{
       ranging_received = ?RANGING_ACTIVE,
       ranging_counter_start = RxStamp,
       ranging_counter_stop = TxStamp,
       ranging_tracking_interval = Rxttcki,
       ranging_offset = Rxtofs,
       ranging_FOM = <<0:8>>
      };
rng_infos(?DISABLED, _, _, _, _) ->
    #ranging_informations{ranging_received = ?NO_RANGING_REQUESTED}.
```

```erlang
-module(ieee802154_pib).

-export([init/1]).
-export([get/2]).
-export([set/3]).
-export([reset/1]).

-include("ieee802154_pib.hrl").

%--- API ----------------------------------------------------------------------

init(PhyMod) ->
    {PhyMod, default_attributes()}.
```

```erlang

-spec get(State, Attribute) -> Result  when
    State       :: pib_state(),
    Attribute   :: pib_attribute(),
    Result      :: Value | {error, unsupported_attribute},
    Value       :: term().
get({_, Attributes}, Attribute) when is_map_key(Attribute, Attributes) ->
    maps:get(Attribute, Attributes);
get(_, _) ->
    {error, unsupported_attribute}.

-spec set(State, Attribute, Value) -> Results when
    State       :: {PhyMod, Attributes},
    PhyMod      :: module(),
    Attributes  :: pib_attributes(),
    Attribute   :: pib_attribute(),
    Value       :: term(),
    Results     :: {ok, NewState} | {error, NewState, Error},
    NewState    :: pib_state(),
    Error       :: pib_set_error().
set({PhyMod, Attributes}, mac_extended_address, Value) ->
    PhyMod:write(eui, #{eui => Value}), % TODO check the range/type/value given
    {ok, {PhyMod, Attributes#{mac_extended_address => Value}}};
set({PhyMod, Attributes}, mac_short_address, Value) ->
    PhyMod:write(panadr, #{short_addr => Value}),
    {ok, {PhyMod, Attributes#{mac_short_address => Value}}};
set({PhyMod, Attributes}, mac_pan_id, Value) ->
    PhyMod:write(panadr, #{pan_id => Value}),
    {ok, {PhyMod, Attributes#{mac_pan_id => Value}}};
set({PhyMod, Attributes}, Attribute, Value)
  when is_map_key(Attribute, Attributes) ->
    NewAttributes = maps:update(Attribute, Value, Attributes),
    {ok, {PhyMod, NewAttributes}};
set(State, _, _) ->
    % TODO detect if PIB is a read only attribute
    {error, State, unsupported_attribute}.

reset({PhyMod, _}) ->
    {PhyMod, default_attributes()}.

%--- Internal ---------------------------------------------------------------
default_attributes() ->
    #{
      cw0 => 2, % cf. p.22 standard
      mac_extended_address => <<16#FFFFFFFF00000000:64>>,
      % mac_max_BE => 8,
      mac_max_BE => 5,
      mac_max_csma_backoffs => 4,
      % mac_min_BE => 5,
      mac_min_BE => 3,
      mac_pan_id => <<16#FFFF:16>>,
      mac_short_address => <<16#FFFF:16>>
     }.
```

```erlang
-module(ieee802154_pib).

-export([init/1]).
-export([get/2]).
-export([set/3]).
-export([reset/1]).

```

```
-include("ieee802154_pib.hrl").

%--- API -------------------------------------------------------------

init(PhyMod) ->
    {PhyMod, default_attributes()}.

-spec get(State, Attribute) -> Result  when
    State       :: pib_state(),
    Attribute   :: pib_attribute(),
    Result      :: Value | {error, unsupported_attribute},
    Value       :: term().
get({_, Attributes}, Attribute) when is_map_key(Attribute, Attributes) ->
    maps:get(Attribute, Attributes);
get(_, _) ->
    {error, unsupported_attribute}.

-spec set(State, Attribute, Value) -> Results when
    State       :: {PhyMod, Attributes},
    PhyMod      :: module(),
    Attributes  :: pib_attributes(),
    Attribute   :: pib_attribute(),
    Value       :: term(),
    Results     :: {ok, NewState} | {error, NewState, Error},
    NewState    :: pib_state(),
    Error       :: pib_set_error().
set({PhyMod, Attributes}, mac_extended_address, Value) ->
    PhyMod:write(eui, #{eui => Value}), % TODO check the range/type/value given
    {ok, {PhyMod, Attributes#{mac_extended_address => Value}}};
set({PhyMod, Attributes}, mac_short_address, Value) ->
    PhyMod:write(panadr, #{short_addr => Value}),
    {ok, {PhyMod, Attributes#{mac_short_address => Value}}};
set({PhyMod, Attributes}, mac_pan_id, Value) ->
    PhyMod:write(panadr, #{pan_id => Value}),
    {ok, {PhyMod, Attributes#{mac_pan_id => Value}}};
set({PhyMod, Attributes}, Attribute, Value)
  when is_map_key(Attribute, Attributes) ->
    NewAttributes = maps:update(Attribute, Value, Attributes),
    {ok, {PhyMod, NewAttributes}};
set(State, _, _) ->
    % TODO detect if PIB is a read only attribute
    {error, State, unsupported_attribute}.

reset({PhyMod, _}) ->
    {PhyMod, default_attributes()}.

%--- Internal ---------------------------------------------------------
default_attributes() ->
    #{
      cw0 => 2, % cf. p.22 standard
      mac_extended_address => <<16#FFFFFFFF00000000:64>>,
      % mac_max_BE => 8,
      mac_max_BE => 5,
      mac_max_csma_backoffs => 4,
      % mac_min_BE => 5,
      mac_min_BE => 3,
      mac_pan_id => <<16#FFFF:16>>,
      mac_short_address => <<16#FFFF:16>>
     }.
```

```
-module(ieee802154_utils).
```

```

%--- Export -------------------------------------------------------------

-export([pckt_duration/2]).
-export([t_dsym/1]).
-export([symbols_to_usec/2]).

%--- Include ------------------------------------------------------------

-include("pmod_uwb.hrl").

%--- Macros -------------------------------------------------------------

-define(N_PHR, 19).
-define(T_D_SYM_1M, 1025.64).

%--- API ----------------------------------------------------------------

% @doc get the packet duration in ns
pckt_duration(PcktSize, Conf) ->
    #phy_cfg{prf = PRF, psr = PSR, sfd = SFD} = Conf,
    TShr = t_psym(PRF) * (PSR + SFD),
    TPhr = ?N_PHR * ?T_D_SYM_1M,
    TDsym = t_dsym(Conf),
    TPsdu = TDsym * PcktSize * 8,
    TShr + TPhr + TPsdu.

t_psym(16) ->
    993.6;
t_psym(64) ->
    1017.6;
t_psym(PRF) ->
    error({non_supported_prf, PRF}).

% t_dsym according to values of table 99 Std. IEEE.802.15.4
% Over all the possible values supported for the PRF and the channels
% only the bit rate determines T_dsym
t_dsym(#phy_cfg{data_rate = ?DATA_RATE_11KHZ}) ->
    8205.13;
t_dsym(#phy_cfg{data_rate = ?DATA_RATE_84KHZ}) ->
    1025.64;
t_dsym(#phy_cfg{data_rate = ?DATA_RATE_6MHZ}) ->
    128.21;
t_dsym(BitRate) ->
    error({non_supported_bit_rate, BitRate}).

% @doc Converts a value in symbols to usec
% It uses t_psym to perform the convertion (defined in table 99)
% @end
symbols_to_usec(Symbols, #phy_cfg{prf = PRF}) ->
    Symbols * t_psym(PRF) / 1000.
```

```
-module(mac_frame).

-include("mac_frame.hrl").

-export([encode/2]).
-export([encode/3]).
-export([encode_ack/2]).
-export([decode/1]).

```

```
%-----------------------------------------------------------------------------
% @doc builds a mac frame without a payload
% @equiv encode(FrameControl, MacHeader, <<>>)
% @end
%-----------------------------------------------------------------------------
-spec encode(FrameControl :: #frame_control{}, MacHeader :: #mac_header{}) ->
    bitstring().
encode(FrameControl, MacHeader) ->
    encode(FrameControl, MacHeader, <<>>).

%-----------------------------------------------------------------------------
% @doc builds a mac frame
% @returns a MAC frame ready to be transmitted in a bitstring (not including the
    CRC automatically added by the DW1000)
% @end
%-----------------------------------------------------------------------------
-spec encode(
    FrameControl :: frame_control(),
    MacHeader :: mac_header(),
    Payload :: bitstring()
) ->
    bitstring().
encode(FrameControl, MacHeader, Payload) ->
    Header = build_mac_header(FrameControl, MacHeader),
    <<Header/bitstring, Payload/bitstring>>.

%-----------------------------------------------------------------------------
% @doc Builds an ACK frame
% @returns a MAC frame ready to be transmitted in a bitstring (not including the
    CRC automatically added by the DW1000)
% @end
%-----------------------------------------------------------------------------
encode_ack(FramePending, Seqnum) ->
    FC = build_frame_control(#frame_control{
        frame_type = ?FTYPE_ACK,
        frame_pending = FramePending,
        dest_addr_mode = ?NONE,
        src_addr_mode = ?NONE
    }),
    <<FC/bitstring, Seqnum:8>>.

%-----------------------------------------------------------------------------
% @doc builds a mac header based on the FrameControl and the MacHeader structures
    given in the args.
% <b> The MAC header doesn't support security fields yet </b>
% @returns the MAC header in a bitstring
% @end
%-----------------------------------------------------------------------------
-spec build_mac_header(FrameControl, MacHeader) -> binary() when
    FrameControl :: frame_control(),
    MacHeader :: mac_header().
build_mac_header(FrameControl, MacHeader) ->
    FC = build_frame_control(FrameControl),

    DestPan = reverse_byte_order(MacHeader#mac_header.dest_pan),
    DestAddr = reverse_byte_order(MacHeader#mac_header.dest_addr),
    DestAddrFields =
        case FrameControl#frame_control.dest_addr_mode of
            ?NONE ->
                <<>>;
            _ ->
                <<DestPan/bitstring, DestAddr/bitstring>>
```

```
        end,

    SrcPan = reverse_byte_order(MacHeader#mac_header.src_pan),
    SrcAddr = reverse_byte_order(MacHeader#mac_header.src_addr),
    SrcAddrFields =
        case
            {
                FrameControl#frame_control.src_addr_mode,
                FrameControl#frame_control.pan_id_compr,
                FrameControl#frame_control.dest_addr_mode
            }
        of
            {?NONE, _, _} ->
                <<>>;
            {_, ?DISABLED, _} ->
                <<SrcPan/bitstring,
                    % if no compression is applied on PANID and SRC addr is present
                    SrcAddr/bitstring>>;
            {_, ?ENABLED, ?NONE} ->
                <<SrcPan/bitstring,
                    % if there is a compression of the PANID but the dest addr isn't present
                    SrcAddr/bitstring>>;
            {_, ?ENABLED, _} ->
                % if there is a compression of the PANID and the dest addr is present
                <<SrcAddr/bitstring>>
        end,
    <<FC/bitstring, (MacHeader#mac_header.seqnum):8, DestAddrFields/bitstring, SrcAddrFields/bitstring>>.

%-----------------------------------------------------------------------------
% @doc decodes the MAC frame given in the arguments
% @return A tuple containing the decoded frame control, the decoded mac header and the payload
% @end
%-----------------------------------------------------------------------------
-spec decode(Data) -> {FrameControl, MacHeader, Payload} when
    Data :: binary(),
    FrameControl :: frame_control(),
    MacHeader :: mac_header(),
    Payload :: bitstring().
decode(Data) ->
    <<FC:16/bitstring, Seqnum:8, Rest/bitstring>> = Data,
    FrameControl = decode_frame_control(FC),
    decode_rest(FrameControl, Seqnum, Rest).

%-----------------------------------------------------------------------------
% @private
% @doc Decodes the remaining sequence of bit present in the payload after the seqnum
% @end
%-----------------------------------------------------------------------------
-spec decode_rest(
    FrameControl :: frame_control(),
    Seqnum :: integer(),
    Rest :: binary()
) ->
    {FrameControl :: frame_control(), MacHeader :: mac_header(), Payload :: binary()}.
decode_rest(
```

```
    #frame_control{frame_type = ?FTYPE_ACK} = FrameControl,
    Seqnum,
    % Might cause an issue if piggybacking is used (allowed in IEEE 802.15.4?)
    _Rest
) ->
    {FrameControl, #mac_header{seqnum = Seqnum}, <<>>};
decode_rest(FrameControl, Seqnum, Rest) ->
    {DestPAN, DestAddr, SrcPAN, SrcAddr, Payload} =
        decode_mac_header(
            FrameControl#frame_control.dest_addr_mode,
            FrameControl#frame_control.src_addr_mode,
            FrameControl#frame_control.pan_id_compr,
            Rest
        ),
    MacHeader =
        #mac_header{
            seqnum = Seqnum,
            dest_pan = reverse_byte_order(DestPAN),
            dest_addr = reverse_byte_order(DestAddr),
            src_pan = reverse_byte_order(SrcPAN),
            src_addr = reverse_byte_order(SrcAddr)
        },
    {FrameControl, MacHeader, Payload}.

% Note extended addresses and PAN ID are used in the case of inter-PAN
    communication
% In inter PAN communication, it can be omitted but it's not mandatory
-spec decode_mac_header(DestAddrMode, SrcAddrMode, PanIdCompr, Bits) ->
    {DestPAN, DestAddr, SrcPAN, SrcAddr, Payload}
when
    DestAddrMode :: flag(),
    SrcAddrMode :: flag(),
    PanIdCompr :: flag(),
    Bits :: bitstring(),
    DestPAN :: binary(),
    DestAddr :: binary(),
    SrcPAN :: binary(),
    SrcAddr :: binary(),
    Payload :: binary().
decode_mac_header(
    ?EXTENDED,
    ?EXTENDED,
    ?DISABLED,
    <<DestPAN:16/bitstring, DestAddr:64/bitstring, SrcPAN:16/bitstring, SrcAddr
        :64/bitstring, Payload/bitstring>>
) ->
    {DestPAN, DestAddr, SrcPAN, SrcAddr, Payload};
decode_mac_header(
    ?EXTENDED,
    ?EXTENDED,
    ?ENABLED,
    <<DestPAN:16/bitstring, DestAddr:64/bitstring, SrcAddr:64/bitstring, Payload/
        bitstring>>
) ->
    {DestPAN, DestAddr, DestPAN, SrcAddr, Payload};
decode_mac_header(
    ?EXTENDED,
    ?SHORT_ADDR,
    ?DISABLED,
    <<DestPAN:16/bitstring, DestAddr:64/bitstring, SrcPAN:16/bitstring, SrcAddr
        :16/bitstring, Payload/bitstring>>
) ->
```

```
    {DestPAN, DestAddr, SrcPAN, SrcAddr, Payload};
decode_mac_header(
    ?EXTENDED,
    ?SHORT_ADDR,
    ?ENABLED,
    <<DestPAN:16/bitstring, DestAddr:64/bitstring, SrcAddr:16/bitstring, Payload/
        bitstring>>
) ->
    {DestPAN, DestAddr, DestPAN, SrcAddr, Payload};
decode_mac_header(
    ?EXTENDED,
    ?NONE,
    _,
    <<DestPAN:16/bitstring, DestAddr:64/bitstring, Payload/bitstring>>
) ->
    {DestPAN, DestAddr, <<>>, <<>>, Payload};
decode_mac_header(
    ?SHORT_ADDR,
    ?EXTENDED,
    ?DISABLED,
    <<DestPAN:16/bitstring, DestAddr:16/bitstring, SrcPAN:16/bitstring, SrcAddr
        :64/bitstring, Payload/bitstring>>
) ->
    {DestPAN, DestAddr, SrcPAN, SrcAddr, Payload};
decode_mac_header(
    ?SHORT_ADDR,
    ?EXTENDED,
    ?ENABLED,
    <<DestPAN:16/bitstring, DestAddr:16/bitstring, SrcAddr:64/bitstring, Payload/
        bitstring>>
) ->
    {DestPAN, DestAddr, DestPAN, SrcAddr, Payload};
decode_mac_header(
    ?SHORT_ADDR,
    ?SHORT_ADDR,
    ?DISABLED,
    <<DestPAN:16/bitstring, DestAddr:16/bitstring, SrcPAN:16/bitstring, SrcAddr
        :16/bitstring, Payload/bitstring>>
) ->
    {DestPAN, DestAddr, SrcPAN, SrcAddr, Payload};
decode_mac_header(
    ?SHORT_ADDR,
    ?SHORT_ADDR,
    ?ENABLED,
    <<DestPAN:16/bitstring, DestAddr:16/bitstring, SrcAddr:16/bitstring, Payload/
        bitstring>>
) ->
    {DestPAN, DestAddr, DestPAN, SrcAddr, Payload};
decode_mac_header(
    ?SHORT_ADDR,
    ?NONE,
    _,
    <<DestPAN:16/bitstring, DestAddr:16/bitstring, Payload/bitstring>>
) ->
    {DestPAN, DestAddr, <<>>, <<>>, Payload};
decode_mac_header(
    ?NONE,
    ?EXTENDED,
    _,
    <<SrcPAN:16/bitstring, SrcAddr:64/bitstring, Payload/bitstring>>
) ->
    {<<>>, <<>>, SrcPAN, SrcAddr, Payload};
```

```
decode_mac_header(
    ?NONE,
    ?SHORT_ADDR,
    _,
    <<SrcPAN:16/bitstring, SrcAddr:16/bitstring, Payload/bitstring>>
) ->
    {<<>>, <<>>, SrcPAN, SrcAddr, Payload};
decode_mac_header(_SrcAddrMode, _DestAddrMode, _PanIdCompr, _Bits) ->
    error(internal_decoding_error).

%-----------------------------------------------------------------------------
% @private
% @doc Creates a MAC frame control
% @param FrameType: MAC frame type
% @param AR: ACK request
% @end
%-----------------------------------------------------------------------------
-spec build_frame_control(FrameControl) -> <<_:16>> when FrameControl ::
    frame_control().
build_frame_control(FrameControl) ->
    #frame_control{
        pan_id_compr = PanIdCompr,
        ack_req = AckReq,
        frame_pending = FramePending,
        sec_en = SecEn,
        frame_type = FrameType,
        src_addr_mode = SrcAddrMode,
        frame_version = FrameVersion,
        dest_addr_mode = DestAddrMode
    } =
        FrameControl,
    <<2#0:1, PanIdCompr:1, AckReq:1, FramePending:1, SecEn:1, FrameType:3,
        SrcAddrMode:2, FrameVersion:2, DestAddrMode:2, 2#0:2>>.

%-----------------------------------------------------------------------------
% @private
% @doc Decode the frame control given in a bitstring form in the parameters
% @end
%-----------------------------------------------------------------------------
-spec decode_frame_control(FC) -> frame_control() when FC :: <<_:16>>.
decode_frame_control(FC) ->
    <<_:1, PanIdCompr:1, AckReq:1, FramePending:1, SecEn:1, FrameType:3,
        SrcAddrMode:2, FrameVersion:2, DestAddrMode:2, _:2>> =
        FC,
    #frame_control{
        frame_type = FrameType,
        sec_en = SecEn,
        frame_pending = FramePending,
        ack_req = AckReq,
        pan_id_compr = PanIdCompr,
        dest_addr_mode = DestAddrMode,
        frame_version = FrameVersion,
        src_addr_mode = SrcAddrMode
    }.

%--- Tool functions ----------------------------------------------------------

% reverse_byte_order(Bitstring) ->
%     Size = bit_size(Bitstring),
%     <<X:Size/integer-little>> = Bitstring,
%     <<X:Size/integer-big>>.
reverse_byte_order(Bitstring) ->
```

```
    reverse_byte_order(Bitstring, <<>>).

reverse_byte_order(<<>>, Acc) ->
    Acc;
reverse_byte_order(<<Head:8>>, Acc) ->
    <<Head:8, Acc/bitstring>>;
reverse_byte_order(<<Head:8, Tail/bitstring>>, Acc) ->
    reverse_byte_order(Tail, <<Head:8, Acc/bitstring>>).
```

```
-module(mac_frame).

-include("mac_frame.hrl").

-export([encode/2]).
-export([encode/3]).
-export([encode_ack/2]).
-export([decode/1]).

%-------------------------------------------------------------------------------
% @doc builds a mac frame without a payload
% @equiv encode(FrameControl, MacHeader, <<>>)
% @end
%-------------------------------------------------------------------------------
-spec encode(FrameControl :: #frame_control{}, MacHeader :: #mac_header{}) ->
    bitstring().
encode(FrameControl, MacHeader) ->
    encode(FrameControl, MacHeader, <<>>).

%-------------------------------------------------------------------------------
% @doc builds a mac frame
% @returns a MAC frame ready to be transmitted in a bitstring (not including the
    CRC automatically added by the DW1000)
% @end
%-------------------------------------------------------------------------------
-spec encode(
    FrameControl :: frame_control(),
    MacHeader :: mac_header(),
    Payload :: bitstring()
) ->
    bitstring().
encode(FrameControl, MacHeader, Payload) ->
    Header = build_mac_header(FrameControl, MacHeader),
    <<Header/bitstring, Payload/bitstring>>.

%-------------------------------------------------------------------------------
% @doc Builds an ACK frame
% @returns a MAC frame ready to be transmitted in a bitstring (not including the
    CRC automatically added by the DW1000)
% @end
%-------------------------------------------------------------------------------
encode_ack(FramePending, Seqnum) ->
    FC = build_frame_control(#frame_control{
        frame_type = ?FTYPE_ACK,
        frame_pending = FramePending,
        dest_addr_mode = ?NONE,
        src_addr_mode = ?NONE
    }),
    <<FC/bitstring, Seqnum:8>>.

%-------------------------------------------------------------------------------
% @doc builds a mac header based on the FrameControl and the MacHeader structures
```

```
    given in the args.
% <b> The MAC header doesn't support security fields yet </b>
% @returns the MAC header in a bitstring
% @end
%-------------------------------------------------------------------------------
-spec build_mac_header(FrameControl, MacHeader) -> binary() when
    FrameControl :: frame_control(),
    MacHeader :: mac_header().
build_mac_header(FrameControl, MacHeader) ->
    FC = build_frame_control(FrameControl),

    DestPan = reverse_byte_order(MacHeader#mac_header.dest_pan),
    DestAddr = reverse_byte_order(MacHeader#mac_header.dest_addr),
    DestAddrFields =
        case FrameControl#frame_control.dest_addr_mode of
            ?NONE ->
                <<>>;
            _ ->
                <<DestPan/bitstring, DestAddr/bitstring>>
        end,

    SrcPan = reverse_byte_order(MacHeader#mac_header.src_pan),
    SrcAddr = reverse_byte_order(MacHeader#mac_header.src_addr),
    SrcAddrFields =
        case
            {
                FrameControl#frame_control.src_addr_mode,
                FrameControl#frame_control.pan_id_compr,
                FrameControl#frame_control.dest_addr_mode
            }
        of
            {?NONE, _, _} ->
                <<>>;
            {_, ?DISABLED, _} ->
                <<SrcPan/bitstring,
                    % if no compression is applied on PANID and SRC addr is
                        present
                    SrcAddr/bitstring>>;
            {_, ?ENABLED, ?NONE} ->
                <<SrcPan/bitstring,
                    % if there is a compression of the PANID but the dest addr isn
                        't present
                    SrcAddr/bitstring>>;
            {_, ?ENABLED, _} ->
                % if there is a compression of the PANID and the dest addr is
                    present
                <<SrcAddr/bitstring>>
        end,
    <<FC/bitstring, (MacHeader#mac_header.seqnum):8, DestAddrFields/bitstring,
        SrcAddrFields/bitstring>>.

%-------------------------------------------------------------------------------
% @doc decodes the MAC frame given in the arguments
% @return A tuple containing the decoded frame control, the decoded mac header and
     the payload
% @end
%-------------------------------------------------------------------------------
-spec decode(Data) -> {FrameControl, MacHeader, Payload} when
    Data :: binary(),
    FrameControl :: frame_control(),
    MacHeader :: mac_header(),
    Payload :: bitstring().
```

```
decode(Data) ->
    <<FC:16/bitstring, Seqnum:8, Rest/bitstring>> = Data,
    FrameControl = decode_frame_control(FC),
    decode_rest(FrameControl, Seqnum, Rest).

%-------------------------------------------------------------------------------
% @private
% @doc Decodes the remaining sequence of bit present in the payload after the seqnum
% @end
%-------------------------------------------------------------------------------
-spec decode_rest(
    FrameControl :: frame_control(),
    Seqnum :: integer(),
    Rest :: binary()
) ->
    {FrameControl :: frame_control(), MacHeader :: mac_header(), Payload :: binary()}.
decode_rest(
    #frame_control{frame_type = ?FTYPE_ACK} = FrameControl,
    Seqnum,
    % Might cause an issue if piggybacking is used (allowed in IEEE 802.15.4?)
    _Rest
) ->
    {FrameControl, #mac_header{seqnum = Seqnum}, <<>>};
decode_rest(FrameControl, Seqnum, Rest) ->
    {DestPAN, DestAddr, SrcPAN, SrcAddr, Payload} =
        decode_mac_header(
            FrameControl#frame_control.dest_addr_mode,
            FrameControl#frame_control.src_addr_mode,
            FrameControl#frame_control.pan_id_compr,
            Rest
        ),
    MacHeader =
        #mac_header{
            seqnum = Seqnum,
            dest_pan = reverse_byte_order(DestPAN),
            dest_addr = reverse_byte_order(DestAddr),
            src_pan = reverse_byte_order(SrcPAN),
            src_addr = reverse_byte_order(SrcAddr)
        },
    {FrameControl, MacHeader, Payload}.

% Note extended addresses and PAN ID are used in the case of inter-PAN communication
% In inter PAN communication, it can be omitted but it's not mandatory
-spec decode_mac_header(DestAddrMode, SrcAddrMode, PanIdCompr, Bits) ->
    {DestPAN, DestAddr, SrcPAN, SrcAddr, Payload}
when
    DestAddrMode :: flag(),
    SrcAddrMode :: flag(),
    PanIdCompr :: flag(),
    Bits :: bitstring(),
    DestPAN :: binary(),
    DestAddr :: binary(),
    SrcPAN :: binary(),
    SrcAddr :: binary(),
    Payload :: binary().
decode_mac_header(
    ?EXTENDED,
    ?EXTENDED,
    ?DISABLED,
```

```
    <<DestPAN:16/bitstring, DestAddr:64/bitstring, SrcPAN:16/bitstring, SrcAddr
        :64/bitstring, Payload/bitstring>>
) ->
    {DestPAN, DestAddr, SrcPAN, SrcAddr, Payload};
decode_mac_header(
    ?EXTENDED,
    ?EXTENDED,
    ?ENABLED,
    <<DestPAN:16/bitstring, DestAddr:64/bitstring, SrcAddr:64/bitstring, Payload/
        bitstring>>
) ->
    {DestPAN, DestAddr, DestPAN, SrcAddr, Payload};
decode_mac_header(
    ?EXTENDED,
    ?SHORT_ADDR,
    ?DISABLED,
    <<DestPAN:16/bitstring, DestAddr:64/bitstring, SrcPAN:16/bitstring, SrcAddr
        :16/bitstring, Payload/bitstring>>
) ->
    {DestPAN, DestAddr, SrcPAN, SrcAddr, Payload};
decode_mac_header(
    ?EXTENDED,
    ?SHORT_ADDR,
    ?ENABLED,
    <<DestPAN:16/bitstring, DestAddr:64/bitstring, SrcAddr:16/bitstring, Payload/
        bitstring>>
) ->
    {DestPAN, DestAddr, DestPAN, SrcAddr, Payload};
decode_mac_header(
    ?EXTENDED,
    ?NONE,
    _,
    <<DestPAN:16/bitstring, DestAddr:64/bitstring, Payload/bitstring>>
) ->
    {DestPAN, DestAddr, <<>>, <<>>, Payload};
decode_mac_header(
    ?SHORT_ADDR,
    ?EXTENDED,
    ?DISABLED,
    <<DestPAN:16/bitstring, DestAddr:16/bitstring, SrcPAN:16/bitstring, SrcAddr
        :64/bitstring, Payload/bitstring>>
) ->
    {DestPAN, DestAddr, SrcPAN, SrcAddr, Payload};
decode_mac_header(
    ?SHORT_ADDR,
    ?EXTENDED,
    ?ENABLED,
    <<DestPAN:16/bitstring, DestAddr:16/bitstring, SrcAddr:64/bitstring, Payload/
        bitstring>>
) ->
    {DestPAN, DestAddr, DestPAN, SrcAddr, Payload};
decode_mac_header(
    ?SHORT_ADDR,
    ?SHORT_ADDR,
    ?DISABLED,
    <<DestPAN:16/bitstring, DestAddr:16/bitstring, SrcPAN:16/bitstring, SrcAddr
        :16/bitstring, Payload/bitstring>>
) ->
    {DestPAN, DestAddr, SrcPAN, SrcAddr, Payload};
decode_mac_header(
    ?SHORT_ADDR,
    ?SHORT_ADDR,
```

```
    ?ENABLED,
    <<DestPAN:16/bitstring, DestAddr:16/bitstring, SrcAddr:16/bitstring, Payload/bitstring>>
) ->
    {DestPAN, DestAddr, DestPAN, SrcAddr, Payload};
decode_mac_header(
    ?SHORT_ADDR,
    ?NONE,
    _,
    <<DestPAN:16/bitstring, DestAddr:16/bitstring, Payload/bitstring>>
) ->
    {DestPAN, DestAddr, <<>>, <<>>, Payload};
decode_mac_header(
    ?NONE,
    ?EXTENDED,
    _,
    <<SrcPAN:16/bitstring, SrcAddr:64/bitstring, Payload/bitstring>>
) ->
    {<<>>, <<>>, SrcPAN, SrcAddr, Payload};
decode_mac_header(
    ?NONE,
    ?SHORT_ADDR,
    _,
    <<SrcPAN:16/bitstring, SrcAddr:16/bitstring, Payload/bitstring>>
) ->
    {<<>>, <<>>, SrcPAN, SrcAddr, Payload};
decode_mac_header(_SrcAddrMode, _DestAddrMode, _PanIdCompr, _Bits) ->
    error(internal_decoding_error).

%-------------------------------------------------------------------------------
% @private
% @doc Creates a MAC frame control
% @param FrameType: MAC frame type
% @param AR: ACK request
% @end
%-------------------------------------------------------------------------------
-spec build_frame_control(FrameControl) -> <<_:16>> when FrameControl :: frame_control().
build_frame_control(FrameControl) ->
    #frame_control{
        pan_id_compr = PanIdCompr,
        ack_req = AckReq,
        frame_pending = FramePending,
        sec_en = SecEn,
        frame_type = FrameType,
        src_addr_mode = SrcAddrMode,
        frame_version = FrameVersion,
        dest_addr_mode = DestAddrMode
    } =
        FrameControl,
    <<2#0:1, PanIdCompr:1, AckReq:1, FramePending:1, SecEn:1, FrameType:3, SrcAddrMode:2, FrameVersion:2, DestAddrMode:2, 2#0:2>>.

%-------------------------------------------------------------------------------
% @private
% @doc Decode the frame control given in a bitstring form in the parameters
% @end
%-------------------------------------------------------------------------------
-spec decode_frame_control(FC) -> frame_control() when FC :: <<_:16>>.
decode_frame_control(FC) ->
    <<_:1, PanIdCompr:1, AckReq:1, FramePending:1, SecEn:1, FrameType:3, SrcAddrMode:2, FrameVersion:2, DestAddrMode:2, _:2>> =
```

```
        FC,
    #frame_control{
        frame_type = FrameType,
        sec_en = SecEn,
        frame_pending = FramePending,
        ack_req = AckReq,
        pan_id_compr = PanIdCompr,
        dest_addr_mode = DestAddrMode,
        frame_version = FrameVersion,
        src_addr_mode = SrcAddrMode
    }.

%--- Tool functions ------------------------------------------------------------

% reverse_byte_order(Bitstring) ->
%     Size = bit_size(Bitstring),
%     <<X:Size/integer-little>> = Bitstring,
%     <<X:Size/integer-big>>.
reverse_byte_order(Bitstring) ->
    reverse_byte_order(Bitstring, <<>>).

reverse_byte_order(<<>>, Acc) ->
    Acc;
reverse_byte_order(<<Head:8>>, Acc) ->
    <<Head:8, Acc/bitstring>>;
reverse_byte_order(<<Head:8, Tail/bitstring>>, Acc) ->
    reverse_byte_order(Tail, <<Head:8, Acc/bitstring>>).
```

```
-module(pmod_uwb).
-behaviour(gen_server).

% API
-export([start_link/2]).
-export([read/1, write/2, write_tx_data/1, get_received_data/0, transmit/1,
    transmit/2, wait_for_transmission/0, reception/0, reception/1]).
-export([reception_async/0]).
-export([set_frame_timeout/1]).
-export([set_preamble_timeout/1, disable_preamble_timeout/0]).
-export([softreset/0, clear_rx_flags/0]).
-export([disable_rx/0]).
-export([suspend_frame_filtering/0, resume_frame_filtering/0]).
-export([signal_power/0]).
-export([prf_value/0]).
-export([rx_preamble_repetition/0]).
-export([rx_data_rate/0]).
-export([rx_ranging_info/0]).
-export([std_noise/0]).
-export([first_path_power_level/0]).
-export([get_conf/0]).
-export([get_rx_metadata/0]).

% gen_server callback
-export([init/1, handle_call/3, handle_cast/2]).

-compile({nowarn_unused_function, [debug_read/2, debug_write/2, debug_write/3,
    debug_bitstring/1, debug_bitstring_hex/1]}).

% Includes
-include("grisp.hrl").

-include("pmod_uwb.hrl").
```

```

%--- Macros --------------------------------------------------------------------

% Define the polarity and the phase of the clock
-define(SPI_MODE, #{clock => {low, leading}}).

-define(WRITE_ONLY_REG_FILE(RegFileID), RegFileID == tx_buffer).

-define(READ_ONLY_REG_FILE(RegFileID), RegFileID==dev_id;
                                       RegFileID==sys_time;
                                       RegFileID==rx_finfo;
                                       RegFileID==rx_buffer;
                                       RegFileID==rx_fqual;
                                       RegFileID==rx_ttcko;
                                       RegFileID==rx_time;
                                       RegFileID==tx_time;
                                       RegFileID==sys_state;
                                       RegFileID==acc_mem).

%% The congifurations of the subregisters of these register files are different
%% (some sub-registers are RO, some are RW and some have reserved bytes
%% that can't be written)
%% Thus, some registers files require to write their sub-register independently
%% => Write the sub-registers one by one instead of writting
%%    the whole register file directly
-define(IS_SRW(RegFileID), RegFileID==agc_ctrl;
                           RegFileID==ext_sync;
                           RegFileID==ec_ctrl;
                           RegFileID==gpio_ctrl;
                           RegFileID==drx_conf;
                           RegFileID==rf_conf;
                           RegFileID==tx_cal;
                           RegFileID==fs_ctrl;
                           RegFileID==aon;
                           RegFileID==otp_if;
                           RegFileID==lde_if;
                           RegFileID==dig_diag;
                           RegFileID==pmsc).

-define(READ_ONLY_SUB_REG(SubRegister), SubRegister==irqs;
                                        SubRegister==agc_stat1;
                                        SubRegister==ec_rxtc;
                                        SubRegister==ec_glop;
                                        SubRegister==drx_car_int;
                                        SubRegister==rf_status;
                                        SubRegister==tc_sarl;
                                        SubRegister==sarw;
                                        SubRegister==tc_pg_status;
                                        SubRegister==lde_thresh;
                                        SubRegister==lde_ppindx;
                                        SubRegister==lde_ppampl;
                                        SubRegister==evc_phe;
                                        SubRegister==evc_rse;
                                        SubRegister==evc_fcg;
                                        SubRegister==evc_fce;
                                        SubRegister==evc_ffr;
                                        SubRegister==evc_ovr;
                                        SubRegister==evc_sto;
                                        SubRegister==evc_pto;
                                        SubRegister==evc_fwto;
                                        SubRegister==evc_txfs;
                                        SubRegister==evc_hpw;
```

```
                                                 SubRegister==evc_tpw).


%--- Types --------------------------------------------------------------------
-export_type([register_values/0]).

-type regFileID() :: atom().
-opaque register_values() :: map().

%--- API ----------------------------------------------------------------------

start_link(Connector, _Opts) ->
    gen_server:start_link({local, ?MODULE}, ?MODULE, Connector, []).


%% @doc read a register file
%%
%% === Example ===
%% To read the register file DEV_ID
%% ```
%% 1> pmod_uwb:read(dev_id).
%% #{model => 1,rev => 0,ridtag => "DECA",ver => 3}
%% '''
-spec read(RegFileID) -> Result when
    RegFileID :: regFileID(),
    Result    :: map() | {error, any()}.
read(RegFileID) when ?WRITE_ONLY_REG_FILE(RegFileID) ->
    error({read_on_write_only_register, RegFileID});
read(RegFileID) -> call({read, RegFileID}).

%% @doc Write values in a register
%%
%% === Examples ===
%% To write in a simple register file (i.e. a register without any sub-register)
%% ```
%% 1> pmod_uwb:write(eui, #{eui => <<16#AAAAAABBBBBBBBBB>>}).
%% ok
%% '''
%% To write in one sub-register of a register file:
%% ```
%% 2> pmod_uwb:write(panadr, #{pan_id => <<16#AAAA>>}).
%% ok
%% '''
%% The previous code will only change the values inside the sub-register PAN_ID
%%
%% To write in multiple sub-register of a register file in the same burst:
%% ```
%% 3> pmod_uwb:write(panadr, #{pan_id => <<16#AAAA>>,
%%                             short_addr => <<16#BBBB>>}).
%% ok
%% '''
%% Some sub-registers have their own fields. For example to set the value of
%% the DIS_AM field in the sub-register AGC_CTRL1 of the register file AGC_CTRL:
%% ```
%% 4> pmod_uwb:write(agc_ctrl, #{agc_ctrl1 => #{dis_am => 2#0}}).
%% '''
-spec write(RegFileID, Value) -> Result when
    RegFileID :: regFileID(),
    Value     :: map(),
    Result    :: ok | {error, any()}.
write(RegFileID, Value) when ?READ_ONLY_REG_FILE(RegFileID) ->
    error({write_on_read_only_register, RegFileID, Value});
```

```
write(RegFileID, Value) when is_map(Value) ->
    call({write, RegFileID, Value}).

%% @doc Writes the data in the TX_BUFFER register
%%
%% Value is expected to be a <b>Binary</b>
%% That choice was made to make the transmission of frames easier later on
%%
%% === Examples ===
%% Send "Hello" in the buffer
%% ‘‘‘
%% 1> pmod_uwb:write_tx_data(<<"Hello">>).
%% ’’’
-spec write_tx_data(Value) -> Result when
    Value  :: binary(),
    Result :: ok | {error, any()}.
write_tx_data(Value) -> call({write_tx, Value}).

%% @doc Retrieves the data received on the UWB antenna
%% @returns {DataLength, Data}
-spec get_received_data() -> Result when
    Result :: {integer(), bitstring()} | {error, any()}.
get_received_data() -> call({get_rx_data}).

get_rx_metadata() ->
    #{rng := Rng} = read(rx_finfo),
    #{rx_stamp := RxStamp} = read(rx_time),
    #{tx_stamp := TxStamp} = read(tx_time),
    #{rxtofs := Rxtofs} = read(rx_ttcko),
    #{rxttcki := Rxttcki} = read(rx_ttcki),
    #{snr => snr(),
      prf => prf_value(),
      pre => rx_preamble_repetition(),
      data_rate => rx_data_rate(),
      rng => Rng,
      rx_stamp => RxStamp,
      tx_stamp => TxStamp,
      rxtofs => Rxtofs,
      rxttcki => Rxttcki}.

% Source: https://forum.qorvo.com/t/how-to-calculate-the-signal-to-noise-ratio-snr-of-dw1000/5585/3
snr() ->
    Delta = 87-7.5,
    RSL = pmod_uwb:signal_power(),
    RSL + Delta.

%% @doc Transmit data with the default options (i.e. don't wait for resp, ...)
%%
%% === Examples ===
%% To transmit a frame:
%% ‘‘‘
%% 1> pmod_uwb:transmit(<Version:4, NextHop:8>>).
%% ok.
%% ’’’
-spec transmit(Data) -> Result when
    Data   :: bitstring(),
    Result :: ok.
transmit(Data) when is_bitstring(Data) ->
    call({transmit, Data, #tx_opts{}}),
    wait_for_transmission().
```

```erlang
%% @doc Performs a transmission with the specified options
%%
%% === Options ===
%% * wait4resp: It specifies that the reception must be enabled after
%%              the transmission in the expectation of a response
%% * w4r-tim: Specifies the turn around time in microseconds. That is the time
%%            the pmod will wait before enabling rx after a tx.
%%            Note that it won't be set if wit4resp is disabled
%% * txdlys: Specifies if the transmitter delayed sending should be set
%% * tx_delay: Specifies the delay of the transmission (see register DX_TIME)
%%
%% === Examples ===
%% To transmit a frame with default options:
%% ```
%% 1> pmod_uwb:transmit(<Version:4, NextHop:8>>, #tx_opts{}).
%% ok.
%% '''
-spec transmit(Data, Options) -> Result when
    Data :: bitstring(),
    Options :: tx_opts(),
    Result :: ok.
transmit(Data, Options) ->
    case Options#tx_opts.wait4resp of
        ?ENABLED -> clear_rx_flags();
        _ -> ok
    end,
    call({transmit, Data, Options}),
    case read(sys_status) of
        #{hdpwarn := 2#1} -> error({hdpwarn});
        _ -> ok
    end,
    wait_for_transmission().

%% Wait for the transmission to be performed
%% usefull in the case of a delayed transmission
wait_for_transmission() ->
    case read(sys_status) of
        #{txfrs := 1} -> ok;
        _ -> wait_for_transmission()
    end.

%% @doc Receive data using the pmod
%% @equiv reception(false)
-spec reception() -> Result when
    Result :: {integer(), bitstring()} | {error, any()}.
reception() ->
    reception(false).

%% @doc Receive data using the pmod
%%
%% The function will hang until a frame is received on the board
%%
%% The CRC of the received frame <b>isn't</b> included in the returned value
%%
%% @param RXEnabled: specifies if the reception is already enabled on the board
%%                   (or set with delay)
%%
%% === Example ===
%% ```
%% 1> pmod_uwb:reception().
%% % Some frame is transmitted
%% {11, <<"Hello world">>}.
```

```
%% '''
-spec reception(RXEnabled) -> Result when
      RXEnabled :: boolean(),
      Result    :: {integer(), bitstring()} | {error, any()}.
reception(RXEnabled) ->
    if not RXEnabled -> enable_rx();
       true -> ok
    end,
    case wait_for_reception() of
        ok ->
            get_received_data();
        Err ->
            {error, Err}
    end.

-spec reception_async() -> Result when
      Result    :: ok | {error, any()}.
reception_async() ->
    case reception() of
        {error, _} = Err -> Err;
        Frame ->
            Metadata = get_rx_metadata(),
            ieee802154_events:rx_event(Frame, Metadata)
    end.

%% @private
enable_rx() ->
    % io:format("Enabling reception~n"),
    clear_rx_flags(),
    call({write, sys_ctrl, #{rxenab => 2#1}}).

%% @doc Disables the reception on the pmod
disable_rx() ->
    call({write, sys_ctrl, #{trxoff => 2#1}}).

wait_for_reception() ->
    % io:format("Wait for resp~n"),
    case read(sys_status) of
        #{rxrfto := 1} -> rxrfto;
        #{rxphe := 1} -> rxphe;
        #{rxfce := 1} -> rxfce;
        #{rxrfsl := 1} -> rxrfsl;
        #{rxpto := 1} -> rxpto;
        #{rxsfdto := 1} -> rxsfdto;
        #{ldeerr := 1} -> ldeerr;
        #{affrej := 1} -> affrej;
        #{rxdfr := 0} -> wait_for_reception();
        #{rxfce := 1} -> rxfce;
        #{rxfcg := 1} -> ok;
        #{rxfcg := 0} -> wait_for_reception();
        % #{rxdfr := 1, rxfcg := 1} -> ok; % The example driver doesn't do that but the user manual says that how you should check the reception of a frame
        _ -> error({error_wait_for_reception})
    end.

%% @doc Set the frame wait timeout and enables it
%% The unit is roughly 1us (cf. user manual)
%% If a float is given, it's decimal part is removed using trunc/1
%% @end
-spec set_frame_timeout(Timeout) -> Result when
      Timeout :: microseconds(),
```

```
      Result  :: ok.
set_frame_timeout(Timeout) when is_float(Timeout) ->
    set_frame_timeout(trunc(Timeout));
set_frame_timeout(Timeout) when is_integer(Timeout) ->
    write(rx_fwto, #{rxfwto => Timeout}),
    write(sys_cfg, #{rxwtoe => 2#1}). % enable receive wait timeout

%% @doc Sets the preamble timeout. (PRETOC register of the DW1000)
%% The unit of 'Timeout' is in units usec
%% If the value is a float, trunc is called to remove the decimal part
%% Internally, it's converted in untis of PAC size
-spec set_preamble_timeout(Timeout) -> ok when
      Timeout :: non_neg_integer().
set_preamble_timeout(TO) when is_float(TO) ->
    set_preamble_timeout(trunc(TO));
set_preamble_timeout(TO) when is_integer(TO) ->
    call({preamble_timeout, TO}),
    write(drx_conf, #{drx_pretoc => 0}).

disable_preamble_timeout() ->
    write(drx_conf, #{drx_pretoc => 0}).

%% @doc Performs a reset of the IC following the procedure (cf. sec. 7.2.50.1)
softreset() ->
    write(pmsc, #{pmsc_ctrl0 => #{sysclks => 2#01}}),
    write(pmsc, #{pmsc_ctrl0 => #{softrest => 16#0}}),
    write(pmsc, #{pmsc_ctrl0 => #{softreset => 16#FFFF}}).


clear_rx_flags() ->
    write(sys_status, #{rxsfdto => 2#1,
                        rxpto => 2#1,
                        rxrfto => 2#1,
                        rxrfsl => 2#1,
                        rxfce => 2#1,
                        rxphe => 2#1,
                        rxprd => 2#1,
                        rxdsfdd => 2#1,
                        rxphd => 2#1,
                        rxdfr => 2#1,
                        rxfcg => 2#1}).

suspend_frame_filtering() ->
    write(sys_cfg, #{ffen => 2#0}).

resume_frame_filtering() ->
    write(sys_cfg, #{ffen => 2#1}).

%% @doc Returns the estimated value of the signal power in dBm
%% cf. user manual section 4.7.2
signal_power() ->
    C = channel_impulse_resp_pow() , % Channel impulse resonse power value (
        CIR_PWR)
    A = case prf_value() of
            16 -> 113.77;
            64 -> 121.74
        end, % Constant. For PRF of 16 MHz = 113.77, for PRF of 64MHz = 121.74
    N = preamble_acc(), % Preamble accumulation count value (RXPACC but might be
        ajusted)
    % io:format("C: ~w~n A:~w~n N:~w~n", [C, A, N]),
    Res = 10 * math:log10((C* math:pow(2, 17))/math:pow(N, 2)) - A,
    % io:format("Estimated signal power: ~p dBm~n", [Res]),
```

```
    % io:format("Std noise: ~w~n", [pmod_uwb:read(rx_fqual)]),
    Res.

preamble_acc() ->
    #{rxpacc := RXPACC} = read(rx_finfo),
    #{rxpacc_nosat := RXPACC_NOSAT} = read(drx_conf),
    if
        RXPACC == RXPACC_NOSAT -> RXPACC - 5;
        true -> RXPACC
    end.

channel_impulse_resp_pow() ->
    #{cir_pwr := CIR_PWR} = read(rx_fqual),
    CIR_PWR.

%% @doc Gives the value of the PRF in MHz
-spec prf_value() -> 16 | 64.
prf_value() ->
    #{agc_tune1 := AGC_TUNE1} = read(agc_ctrl),
    case AGC_TUNE1 of
        16#8870 -> 16;
        16#889B -> 64
    end.

%% @doc returns the preamble symbols repetition
rx_preamble_repetition() ->
    #{rxpsr := RXPSR} = read(rx_finfo),
    case RXPSR of
        0 -> 16;
        1 -> 64;
        2 -> 1024;
        3 -> 4096
    end.

%% @doc returns the data rate of the received frame in kbps
rx_data_rate() ->
    #{rxbr := RXBR} = read(rx_finfo),
    case RXBR of
        0 -> 110;
        1 -> 850;
        2 -> 6800
    end.

% @doc returns the value of the ‘Ranging’ bit of the received frame
rx_ranging_info() ->
    #{rng := RNG} = read(rx_finfo),
    RNG.

std_noise() ->
    #{std_noise := STD_NOISE} = read(rx_fqual),
    STD_NOISE.

first_path_power_level() ->
    #{fp_ampl1 := F1} = read(rx_time),
    #{fp_ampl2 := F2, pp_ampl3 := F3} = read(rx_fqual),
    A = 113.77,
    N = preamble_acc(),
    10 * math:log10((math:pow(F1,2) + math:pow(F2, 2) + math:pow(F3, 2))/math:pow(
        N, 2)) - A.

get_conf() ->
    call({get_conf}).
```

```

%--- gen_server Callbacks -----------------------------------------------------

%% @private
init(Slot) ->
    % Verify the slot used
    case {grisp_hw:platform(), Slot} of
        {grisp2, spi2} -> ok;
        {P, S} -> error({incompatible_slot, P, S})
    end,
    grisp_devices:register(Slot, ?MODULE),
    Bus = grisp_spi:open(Slot),
    case verify_id(Bus) of
        ok -> softreset(Bus);
        Val -> error({dev_id_no_match, Val})
    end,
    ldeload(Bus),
    % TODO Merge the next 4 cfg commands into one
    write_default_values(Bus),
    config(Bus),
    setup_sfd(Bus),
    Conf =  #phy_cfg{},
    {ok, #{bus => Bus, conf => Conf}}.

%% @private
handle_call({read, RegFileID}, _From, #{bus := Bus} = State)                ->
    {reply, read_reg(Bus, RegFileID), State};
handle_call({write, RegFileID, Value}, _From, #{bus := Bus} = State)        ->
    {reply, write_reg(Bus, RegFileID, Value), State};
handle_call({write_tx, Value}, _From, #{bus := Bus} = State)                ->
    {reply, write_tx_data(Bus, Value), State};
handle_call({transmit, Data, Options}, _From, #{bus := Bus} = State)        ->
    {reply, tx(Bus, Data, Options), State};
handle_call({delayed_transmit, Data, Delay}, _From, #{bus := Bus} = State) ->
    {reply, delayed_tx(Bus, Data, Delay), State};
handle_call({get_rx_data}, _From, #{bus := Bus} = State)                    ->
    {reply, get_rx_data(Bus), State};
handle_call({get_conf}, _From, #{conf := Conf} = State)                     ->
    {reply, Conf, State};
handle_call({preamble_timeout, TOus}, _From, State)                         ->
    #{bus := Bus, conf := Conf} = State,
    PACSize = Conf#phy_cfg.pac_size,
    case TOus of
        0 ->
            write_reg(Bus, drx_conf, #{drx_pretoc => 0});
        _ ->
            % Remove 1 because DW1000 counter auto. adds 1 (cf. 7.2.40.9 user manual)
            To = math:ceil(TOus / PACSize)-1,
            write_reg(Bus, drx_conf, #{drx_pretoc => round(To)})
    end,
    {reply, ok, State};
handle_call(Request, _From, _State)                                         ->
    error({unknown_call, Request}).

%% @private
handle_cast(Request, _State) -> error({unknown_cast, Request}).

%--- Internal -----------------------------------------------------------------

call(Call) ->
    Dev = grisp_devices:default(?MODULE),
```

```
    gen_server:call(Dev#device.pid, Call).


%% @doc Varify the dev_id register of the pmod
%% @returns ok if the value is correct, otherwise the value read
verify_id(Bus) ->
    #{ridtag := RIDTAG, model := MODEL} = read_reg(Bus, dev_id),
    case {RIDTAG, MODEL} of
        {"DECA", 1} -> ok;
        _ -> {RIDTAG, MODEL}
    end.

%% @private
%% Performs a softreset on the pmod
-spec softreset(Bus::grisp_spi:ref()) -> ok.
softreset(Bus) ->
    write_reg(Bus, pmsc, #{pmsc_ctrl0 => #{sysclks => 2#01}}),
    write_reg(Bus, pmsc, #{pmsc_ctrl0 => #{softrest => 16#0}}),
    write_reg(Bus, pmsc, #{pmsc_ctrl0 => #{softreset => 16#FFFF}}).

%% @private
%% Writes the default values described in section 2.5.5 of the user manual
-spec write_default_values(Bus::grisp_spi:ref()) -> ok.
write_default_values(Bus) ->
    write_reg(Bus, lde_if, #{lde_cfg1 => #{ntm => 16#D}, lde_cfg2 => 16#1607}),
    write_reg(Bus, agc_ctrl, #{agc_tune1 => 16#8870, agc_tune2 => 16#2502A907}),
    write_reg(Bus, drx_conf, #{drx_tune2 => 16#311A002D}),
    write_reg(Bus, tx_power, #{tx_power => 16#0E082848}),
    write_reg(Bus, rf_conf, #{rf_txctrl => 16#001E3FE3}),
    write_reg(Bus, tx_cal, #{tc_pgdelay => 16#B5}),
    write_reg(Bus, fs_ctrl, #{fs_plltune => 16#BE}).

%% @private
config(Bus) ->
    write_reg(Bus, ext_sync, #{ec_ctrl => #{pllldt => 2#1}}),
    %write_reg(Bus, pmsc, #{pmsc_ctrl1 => #{lderune => 2#0}}),
    % Now enable RX and TX leds
    write_reg(Bus, gpio_ctrl, #{gpio_mode => #{msgp2 => 2#01, msgp3 => 2#01}}),
    % Enable RXOK and SFD leds
    write_reg(Bus, gpio_ctrl, #{gpio_mode => #{msgp0 => 2#01, msgp1 => 2#01}}),
    write_reg(Bus, pmsc, #{pmsc_ctrl0 => #{gpdce => 2#1, khzclken => 2#1}}),
    write_reg(Bus, pmsc, #{pmsc_ledc => #{blnken => 2#1}}),
    write_reg(Bus, dig_diag, #{evc_ctrl => #{evc_en => 2#1}}), % enable counting event for debug purposes
    % write_reg(Bus, sys_cfg, #{rxwtoe => 2#1}),
    write_reg(Bus, tx_fctrl, #{txpsr => 2#10}). % Setting preamble symbols to 1024

%% @private
%% Load the microcode from ROM to RAM
%% It follows the steps described in section 2.5.5.10 of the DW1000 user manual
ldeload(Bus) ->
    write_reg(Bus, pmsc, #{pmsc_ctrl0 => #{sysclks => 2#01}}),
    write_reg(Bus, pmsc, #{pmsc_ctrl0 => #{otp => 2#1, res8 => 2#1}}), % Writes 0x0301 in pmsc_ctrl0
    write_reg(Bus, otp_if, #{otp_ctrl => #{ldeload => 2#1}}), % Writes 0x8000 in OTP_CTRL
    timer:sleep(150), % User manual requires a wait of 150 s
    write_reg(Bus, pmsc, #{pmsc_ctrl0 => #{sysclks => 2#0}}), % Writes 0x0200 in pmsc_ctrl0
    write_reg(Bus, pmsc, #{pmsc_ctrl0 => #{res8 => 2#0}}).

%% @private
```

```erlang
%% If no frame is transmitted before AUTOACK, then the SFD isn't properly set
%% (cf. section 5.3.1.2 SFD initialisation)
setup_sfd(Bus) ->
    write_reg(Bus, sys_ctrl, #{txstrt => 2#1, trxoff => 2#1}).

%% @private
%% Transmit the data using UWB
%% @param Options is used to set options about the transmission like a transmission delay, etc.
-spec tx(grisp_spi:ref(), Data :: binary(), Options :: #tx_opts{}) -> ok.
tx(Bus, Data, #tx_opts{wait4resp = Wait4resp, w4r_tim = W4rTim, txdlys = TxDlys, tx_delay = TxDelay, ranging = Ranging}) ->
    % Writing the data that will be sent (w/o CRC)
    DataLength = byte_size(Data) + 2, % DW1000 automatically adds the 2 bytes CRC
    write_tx_data(Bus, Data),
    % Setting the options of the transmission
    case Wait4resp of
        ?ENABLED -> write_reg(Bus, ack_resp_t, #{w4r_tim => W4rTim});
        _ -> ok
    end,
    case TxDlys of
        ?ENABLED -> write_reg(Bus, dx_time, #{dx_time => TxDelay});
        _ -> ok
    end,
    write_reg(Bus, tx_fctrl, #{txboffs => 2#0, tr => Ranging, tflen => DataLength}),
    write_reg(Bus, sys_ctrl, #{txstrt => 2#1, wait4resp => Wait4resp, txdlys => TxDlys}). % start transmission and some options

%% @private
%% Transmit the data with a specified delay using UWB
delayed_tx(Bus, Data, Delay) ->
    write_reg(Bus, dx_time, #{dx_time => Delay}),
    DataLength = byte_size(Data) + 2, % DW1000 automatically adds the 2 bytes CRC
    write_tx_data(Bus, Data),
    write_reg(Bus, tx_fctrl, #{txboffs => 2#0, tflen => DataLength}),
    write_reg(Bus, sys_ctrl, #{txstrt => 2#1, txdlys => 2#1}). % start transmission

%% @private
%% Get the received data (without the CRC bytes) stored in the rx_buffer
get_rx_data(Bus) ->
    #{rxflen := FrameLength} = read_reg(Bus, rx_finfo),
    Frame = read_rx_data(Bus, FrameLength-2), % Remove the CRC bytes
    {FrameLength, Frame}.

%% @private
%% @doc Reverse the byte order of the bitstring given in the argument
%% @param Bin a bitstring
reverse(Bin) -> reverse(Bin, <<>>).
reverse(<<Bin:8>>, Acc) ->
    <<Bin, Acc/binary>>;
reverse(<<Bin:8, Rest/bitstring>>, Acc) ->
    reverse(Rest, <<Bin, Acc/binary>>).

% Source: https://stackoverflow.com/a/43310493
% reverse(Binary) ->
%     Size = bit_size(Binary),
%     <<X:Size/integer-little>> = Binary,
%     <<X:Size/integer-big>>.

%% @private
```

```erlang
%% @doc Creates the header of the SPI transaction between the GRiSP and the pmod
%%
%%  It creates a header of 1 bytes. The header is used in a transaction that will affect
%%  the whole register file (read/write)
%%
%% @param Op an atom (either <i>read</i> or <i>write</i>
%% @param RegFileID an atom representing the register file
%% @returns a formated header of <b>1 byte</b> long as described in the user manual
header(Op, RegFileID) ->
    <<(rw(Op)):1, 2#0:1, (regFile(RegFileID)):6>>.

%% @private
%% @doc Creates the header of the SPI transaction between the GRiSP and the pmod
%%
%%  It creates a header of 2 bytes. The header is used in a transaction that will affect
%%  the whole sub-register (read/write)
%%  Careful: The sub-register needs to be mapped in the hrl file
%%
%% @param Op an atom (either <i>read</i> or <i>write</i>
%% @param RegFileID an atom representing the register file
%% @param SubRegister an atom representing the sub-register
%% @returns a formated header of <b>2 byte</b> long as described in the user manual
header(Op, RegFileID, SubRegister) ->
    case subReg(SubRegister) < 127 of
        true -> header(Op, RegFileID, SubRegister, 2);
        _ -> header(Op, RegFileID, SubRegister, 3)
    end.

header(Op, RegFileID, SubRegister, 2) ->
    << (rw(Op)):1, 2#1:1, (regFile(RegFileID)):6,
        2#0:1, (subReg(SubRegister)):7 >>;
header(Op, RegFileID, SubRegister, 3) ->
    <<_:1, HighOrder:8, LowOrder:7>> = <<(subReg(SubRegister)):16>>,
    << (rw(Op)):1, 2#1:1, (regFile(RegFileID)):6,
       2#1:1, LowOrder:7,
       HighOrder:8>>.

%% @private
%% @doc Read the values stored in a register file
read_reg(Bus, lde_ctrl) -> read_reg(Bus, lde_if);
read_reg(Bus, lde_if) ->
    lists:foldl(fun(Elem, Acc) ->
                    Res = read_sub_reg(Bus, lde_if, Elem),
                    maps:merge(Acc, Res)
                end,
                #{},
                [lde_thresh, lde_cfg1, lde_ppindx, lde_ppampl, lde_rxantd, lde_cfg2, lde_repc]);
read_reg(Bus, RegFileID) ->
    Header = header(read, RegFileID),
    [Resp] = grisp_spi:transfer(Bus, [{?SPI_MODE, Header, 1, regSize(RegFileID)}]),
    % debug_read(RegFileID, Resp),
    reg(decode, RegFileID, Resp).


read_sub_reg(Bus, RegFileID, SubRegister) ->
    Header = header(read, RegFileID, SubRegister),
```

```
    HeaderSize = byte_size(Header),
    % io:format("[HEADER] type ~w - ~w - ~w~n", [HeaderSize, Header, subRegSize(SubRegister)]),
    [Resp] = grisp_spi:transfer(Bus, [{?SPI_MODE, Header, HeaderSize, subRegSize(SubRegister)}]),
    reg(decode, SubRegister, Resp).


%% @doc get the received data
%% @param Length is the total length of the data we are trying to read
read_rx_data(Bus, Length) ->
    Header = header(read, rx_buffer),
    [Resp] = grisp_spi:transfer(Bus, [{?SPI_MODE, Header, 1, Length}]),
    Resp.

% TODO: check that user isn't trying to write reserved bits by passing res, res1, ... in the map fields
%% @doc used to write the values in the map given in the Value argument
-spec write_reg(Bus::grisp_spi:ref(), RegFileID::regFileID(), Value::map()) -> ok.
% Write each sub-register one by one.
% If the user tries to write in a read-only sub-register, an error is thrown
write_reg(Bus, RegFileID, Value) when ?IS_SRW(RegFileID) ->
    maps:map(
        fun(SubRegister, Val) ->
            CurrVal = maps:get(SubRegister, read_reg(Bus, RegFileID)), % ? can the read be done before ? Maybe but not assured that no values changes after a write in the register
            Body = case CurrVal of
                        V when is_map(V) -> reg(encode, SubRegister, maps:merge_with(fun(_Key, _Old, New) -> New end, CurrVal, Val));
                        _ -> reg(encode, SubRegister, #{SubRegister => Val})
                   end,
            Header = header(write, RegFileID, SubRegister),
            % debug_write(RegFileID, SubRegister, Body),
            _ = grisp_spi:transfer(Bus, [{?SPI_MODE, <<Header/binary, Body/binary>>, 2+subRegSize(SubRegister), 0}])
        end,
        Value),
    ok;
write_reg(Bus, RegFileID, Value) ->
    Header = header(write, RegFileID),
    CurrVal = read_reg(Bus, RegFileID),
    ValuesToWrite = maps:merge_with(fun(_Key, _Value1, Value2) -> Value2 end, CurrVal, Value),
    Body = reg(encode, RegFileID, ValuesToWrite),
    % debug_write(RegFileID, Body),
    _ = grisp_spi:transfer(Bus, [{?SPI_MODE, <<Header/binary, Body/binary>>, 1+regSize(RegFileID), 0}]),
    ok.

%% @doc write_tx_data/2 sends data (Value) in the register tx_buffer
%% @param Value is the data to be written. It must be a binary and have a size of maximum 1024 bits
write_tx_data(Bus, Value) when is_binary(Value), (bit_size(Value) < 1025) ->
    Header = header(write, tx_buffer),
    Length = byte_size(Value),
    % debug_write(tx_buffer, Body),
    _ = grisp_spi:transfer(Bus, [{?SPI_MODE, <<Header/binary, Value/binary>>, 1+Length, 0}]),
    ok.

```

```
%---- Register mapping ---------------------------------------------------------

%% @doc Used to either decode the data returned by the pmod or to encode to data that will be sent to the pmod
%%
%% The transmission on the MISO line is done byte by byte starting from the lowest rank byte to the highest rank
%% Example: dev_id value is 0xDECA0130 but 0x3001CADE is transmitted over the MISO line
-spec reg(Type, Register, Val) -> Ret when
      Type     :: encode | decode,
      Register :: regFileID(),
      Val      :: nonempty_binary() | register_values(),
      Ret      :: nonempty_binary() | register_values().
reg(encode, SubRegister, Value) when ?READ_ONLY_SUB_REG(SubRegister) -> error({writing_read_only_sub_register, SubRegister, Value});
reg(decode, dev_id, Resp) ->
    <<
      RIDTAG:16, Model:8, Ver:4, Rev:4
    >> = reverse(Resp),
    #{
        ridtag => integer_to_list(RIDTAG, 16), model => Model, ver => Ver, rev => Rev
    };
reg(decode, eui, Resp) ->
    #{
        eui => reverse(Resp)
    };
reg(encode, eui, Val) ->
    #{
        eui:= EUI
    } = Val,
    reverse(
        EUI
    );
reg(decode, panadr, Resp) ->
    <<
        PanId:16, ShortAddr:16
    >> = reverse(Resp),
    #{
        pan_id => <<PanId:16>>, short_addr => <<ShortAddr:16>>
    };
reg(encode, panadr, Val) ->
    #{
        pan_id := PanId, short_addr := ShortAddr
    } = Val,
    reverse(<<
        PanId:16/bitstring, ShortAddr:16/bitstring
    >>);
reg(decode, sys_cfg, Resp) ->
    <<
        FFA4:1, FFAR:1, FFAM:1, FFAA:1, FFAD:1, FFAB:1, FFBC:1, FFEN:1, % bits 7-0
        FCS_INIT2F:1, DIS_RSDE:1, DIS_PHE:1, DIS_DRXB:1, DIS_FCE:1, SPI_EDGE:1, HIRQ_POL:1, FFA5:1, % bits 15-8
        _:1, RXM110K:1, _:3, DIS_STXP:1, PHR_MODE:2, % bits 23-16
        AACKPEND:1, AUTOACK:1, RXAUTR:1, RXWTOE:1, _:4 % bits 31-24
    >> = Resp,
    #{
        aackpend => AACKPEND, autoack => AUTOACK, rxautr => RXAUTR, rxwtoe => RXWTOE,
        rxm110k => RXM110K, dis_stxp => DIS_STXP, phr_mode => PHR_MODE,
        fcs_init2F => FCS_INIT2F, dis_rsde => DIS_RSDE, dis_phe => DIS_PHE,
```

```
            dis_drxb => DIS_DRXB, dis_fce => DIS_FCE, spi_edge => SPI_EDGE,
            hirq_pol => HIRQ_POL, ffa5 => FFA5,
        ffa4 => FFA4, ffar => FFAR, ffam => FFAM, ffaa => FFAA, ffad => FFAD, ffab
             => FFAB, ffbc => FFBC, ffen => FFEN
    };
reg(encode, sys_cfg, Val) ->
    #{
        aackpend := AACKPEND, autoack := AUTOACK, rxautr := RXAUTR, rxwtoe :=
            RXWTOE,
        rxm110k := RXM110K, dis_stxp := DIS_STXP, phr_mode := PHR_MODE,
        fcs_init2F := FCS_INIT2F, dis_rsde := DIS_RSDE, dis_phe := DIS_PHE,
            dis_drxb := DIS_DRXB, dis_fce := DIS_FCE, spi_edge := SPI_EDGE,
            hirq_pol := HIRQ_POL, ffa5 := FFA5,
        ffa4 := FFA4, ffar := FFAR, ffam := FFAM, ffaa := FFAA, ffad := FFAD, ffab
             := FFAB, ffbc := FFBC, ffen := FFEN
    } = Val,
    <<
        FFA4:1, FFAR:1, FFAM:1, FFAA:1, FFAD:1, FFAB:1, FFBC:1, FFEN:1, % bits 7-0
        FCS_INIT2F:1, DIS_RSDE:1, DIS_PHE:1, DIS_DRXB:1, DIS_FCE:1, SPI_EDGE:1,
            HIRQ_POL:1, FFA5:1, % bits 15-8
        2#0:1, RXM110K:1, 2#0:3, DIS_STXP:1, PHR_MODE:2, % bits 23-16
        AACKPEND:1, AUTOACK:1, RXAUTR:1, RXWTOE:1, 2#0:4 % bits 31-24
    >>;
reg(decode, sys_time, Resp) ->
    <<
        SysTime:40
    >> = reverse(Resp),
    #{
        sys_time => SysTime
    };
reg(decode, tx_fctrl, Resp) ->
    <<
        IFSDELAY:8, TXBOFFS:10, PE:2, TXPSR:2, TXPRF:2, TR:1, TXBR:2, R:3, TFLE:3,
             TFLEN:7
    >> = reverse(Resp),
    #{
        ifsdelay => IFSDELAY, txboffs => TXBOFFS, pe => PE, txpsr => TXPSR, txprf
            => TXPRF, tr => TR, txbr => TXBR, r => R, tfle => TFLE, tflen => TFLEN
    };
reg(encode, tx_fctrl, Val) ->
    #{
        ifsdelay := IFSDELAY, txboffs := TXBOFFS, pe := PE, txpsr := TXPSR, txprf
            := TXPRF, tr := TR, txbr := TXBR, r := R, tfle := TFLE, tflen := TFLEN
    } = Val,
    reverse(<<
        IFSDELAY:8, TXBOFFS:10, PE:2, TXPSR:2, TXPRF:2, TR:1, TXBR:2, R:3, TFLE:3,
             TFLEN:7
    >>);
% TX_BUFFER is write only => no decode
reg(decode, dx_time, Resp) ->
    #{
        dx_time => reverse(Resp)
    };
reg(encode, dx_time, Val) ->
    #{
        dx_time := DX_TIME
    } = Val,
    reverse(<<
        DX_TIME:40
    >>);
reg(decode, rx_fwto, Resp) ->
    <<
```

```
        RXFWTO:16
    >> = reverse(Resp),
    #{
        rxfwto => RXFWTO
    };
reg(encode, rx_fwto, Val) ->
    #{
        rxfwto := RXFWTO
    } = Val,
    reverse(<<
        RXFWTO:16
    >>);
reg(decode, sys_ctrl, Resp) ->
    <<
        WAIT4RESP:1, TRXOFF:1, _:2, CANSFCS:1, TXDLYS:1, TXSTRT:1, SFCST:1, % bits 7-0
        _:6, RXDLYE:1, RXENAB:1, % bits 15-8
        _:8, % bits 23-16
        _:7, HRBPT:1 % bits 31-24
    >> = Resp,
    #{
        sfcst => SFCST, txstrt => TXSTRT, txdlys => TXDLYS, cansfcs => CANSFCS, trxoff => TRXOFF, wait4resp => WAIT4RESP,
        rxenab => RXENAB, rxdlye => RXDLYE,
        hrbpt => HRBPT
    };
reg(encode, sys_ctrl, Val) ->
    #{
        sfcst := SFCST, txstrt := TXSTRT, txdlys := TXDLYS, cansfcs := CANSFCS, trxoff := TRXOFF, wait4resp := WAIT4RESP,
        rxenab := RXENAB, rxdlye := RXDLYE,
        hrbpt := HRBPT
    } = Val,
    <<
        WAIT4RESP:1, TRXOFF:1, 2#0:2, CANSFCS:1, TXDLYS:1, TXSTRT:1, SFCST:1, % bits 7-0
        2#0:6, RXDLYE:1, RXENAB:1, % bits 15-8
        2#0:8, % bits 23-16
        2#0:7, HRBPT:1 % bits 31-24
    >>;
reg(decode, sys_mask, Resp) ->
    <<
        MTXFRS:1, MTXPHS:1, MTXPRS:1, MTXFRB:1, MAAT:1, MESYNCR:1, MCPLOCK:1, Reserved0:1, % bits 7-0
        MRXFCE:1, MRXFCG:1, MRXDFR:1, MRXPHE:1, MRXPHD:1, MLDEDON:1, MRXSFDD:1, MRXPRD:1, % bits 15-8
        MSLP2INIT:1, MGPIOIRQ:1, MRXPTO:1, MRXOVRR:1, Reserved1:1, MLDEERR:1, MRXRFTO:1, MRXRFSL:1, % bits 23-16
        Reserved2:2, MAFFREJ:1, MTXBERR:1, MHPDDWAR:1, MPLLHILO:1, MCPLLLL:1, MRFPLLLL:1 % bits 31-24
    >> = Resp,
    #{
        mtxfrs => MTXFRS, mtxphs => MTXPHS, mtxprs => MTXPRS, mtxfrb => MTXFRB, maat => MAAT, mesyncr => MESYNCR, mcplock => MCPLOCK, res0 => Reserved0, % bits 7-0
        mrxfce => MRXFCE, mrxfcg => MRXFCG, mrxdfr => MRXDFR, mrxphe => MRXPHE, mrxphd => MRXPHD, mldeon => MLDEDON, mrxsfdd => MRXSFDD, mrxprd => MRXPRD, % bits 15-8
        mslp2init => MSLP2INIT, mgpioirq => MGPIOIRQ, mrxpto => MRXPTO, mrxovrr => MRXOVRR, res1 => Reserved1, mldeerr => MLDEERR, mrxrfto => MRXRFTO, mrxrfsl => MRXRFSL, % bits 23-16
        res2 => Reserved2, maffrej => MAFFREJ, mtxberr => MTXBERR, mhpddwar =>
```

```
            MHPDDWAR, mpllhilo => MPLLHILO, mcplll1 => MCPLLLL, mrfplll1 =>
            MRFPLLLL % bits 31-24
    };
reg(encode, sys_mask, Val) ->
    #{
        mtxfrs := MTXFRS, mtxphs := MTXPHS, mtxprs := MTXPRS, mtxfrb := MTXFRB,
            maat := MAAT, mesyncr := MESYNCR, mcplock := MCPLOCK, res0 :=
            Reserved0, % bits 7-0
        mrxfce := MRXFCE, mrxfcg := MRXFCG, mrxdfr := MRXDFR, mrxphe := MRXPHE,
            mrxphd := MRXPHD, mldeon := MLDEDON, mrxsfdd := MRXSFDD, mrxprd :=
            MRXPRD, % bits 15-8
        mslp2init := MSLP2INIT, mgpioirq := MGPIOIRQ, mrxpto := MRXPTO, mrxovrr :=
             MRXOVRR, res1 := Reserved1, mldeerr := MLDEERR, mrxrfto := MRXRFTO,
            mrxrfsl := MRXRFSL, % bits 23-16
        res2 := Reserved2, maffrej := MAFFREJ, mtxberr := MTXBERR, mhpddwar :=
            MHPDDWAR, mpllhilo := MPLLHILO, mcplll1 := MCPLLLL, mrfplll1 :=
            MRFPLLLL % bits 31-24
    } = Val,
    <<
        MTXFRS:1, MTXPHS:1, MTXPRS:1, MTXFRB:1, MAAT:1, MESYNCR:1, MCPLOCK:1,
            Reserved0:1, % bits 7-0
        MRXFCE:1, MRXFCG:1, MRXDFR:1, MRXPHE:1, MRXPHD:1, MLDEDON:1, MRXSFDD:1,
            MRXPRD:1, % bits 15-8
        MSLP2INIT:1, MGPIOIRQ:1, MRXPTO:1, MRXOVRR:1, Reserved1:1, MLDEERR:1,
            MRXRFTO:1, MRXRFSL:1, % bits 23-16
        Reserved2:2, MAFFREJ:1, MTXBERR:1, MHPDDWAR:1, MPLLHILO:1, MCPLLLL:1,
            MRFPLLLL:1 % bits 31-24
    >>;
reg(decode, sys_status, Resp) ->
    <<
        TXFRS:1, TXPHS:1, TXPRS:1, TXFRB:1, AAT:1, ESYNCR:1, CPLOCK:1, IRQS:1, %
            bits 7-0
        RXFCE:1, RXFCG:1, RXDFR:1, RXPHE:1, RXPHD:1, LDEDONE:1, RXSFDD:1, RXPRD:1,
             % bits 15-8
        SPL2INIT:1, GPIOIRQ:1, RXPTO:1, RXOVRR:1, Reserved0:1, LDEERR:1, RXRFTO:1,
             RXRFSL:1, % bits 23-16
        ICRBP:1, HSRBP:1, AFFREJ:1, TXBERR:1, HPDWARN:1, RXSFDTO:1, CLCKPLL_LL:1,
            RFPLL_LL:1, % bits 31-24
        Reserved1:5, TXPUTE:1, RXPREJ:1, RXRSCS:1 % bits 39-32
    >> = Resp,
    #{
        txfrs => TXFRS, txphs => TXPHS, txprs => TXPRS, txfrb => TXFRB, aat => AAT
            , esyncr => ESYNCR, cplock => CPLOCK, irqs => IRQS, % bits 7-0
        rxfce => RXFCE, rxfcg => RXFCG, rxdfr => RXDFR, rxphe => RXPHE, rxphd =>
            RXPHD, ldedone => LDEDONE, rxsfdd => RXSFDD, rxprd => RXPRD, % bits
            15-8
        splt2init => SPL2INIT, gpioirq => GPIOIRQ, rxpto => RXPTO, rxovrr =>
            RXOVRR, res0 => Reserved0, ldeerr => LDEERR, rxrfto => RXRFTO, rxrfsl
            => RXRFSL, % bits 23-16
        icrbp => ICRBP, hsrbp => HSRBP, affrej => AFFREJ, txberr => TXBERR,
            hdpwarn => HPDWARN, rxsfdto => RXSFDTO, clkpll_ll => CLCKPLL_LL,
            rfpll_ll => RFPLL_LL, % bits 31-24
        res1 => Reserved1, txpute => TXPUTE, rxprej => RXPREJ, rxrscs => RXRSCS
    };
reg(encode, sys_status, Val) ->
    #{
        txfrs := TXFRS, txphs := TXPHS, txprs := TXPRS, txfrb := TXFRB, aat := AAT
            , esyncr := ESYNCR, cplock := CPLOCK, irqs := IRQS, % bits 7-0
        rxfce := RXFCE, rxfcg := RXFCG, rxdfr := RXDFR, rxphe := RXPHE, rxphd :=
            RXPHD, ldedone := LDEDONE, rxsfdd := RXSFDD, rxprd := RXPRD, % bits
            15-8
        splt2init := SPL2INIT, gpioirq := GPIOIRQ, rxpto := RXPTO, rxovrr :=
```

```
            RXOVRR, res0 := Reserved0, ldeerr := LDEERR, rxrfto := RXRFTO, rxrfsl
            := RXRFSL, % bits 23-16
        icrbp := ICRBP, hsrbp := HSRBP, affrej := AFFREJ, txberr := TXBERR,
            hdpwarn := HPDWARN, rxsfdto := RXSFDTO, clkpll_ll := CLCKPLL_LL,
            rfpll_ll := RFPLL_LL, % bits 31-24
        res1 := Reserved1, txpute := TXPUTE, rxprej := RXPREJ, rxrscs := RXRSCS
    } = Val,
    <<
        TXFRS:1, TXPHS:1, TXPRS:1, TXFRB:1, AAT:1, ESYNCR:1, CPLOCK:1, IRQS:1, %
            bits 7-0
        RXFCE:1, RXFCG:1, RXDFR:1, RXPHE:1, RXPHD:1, LDEDONE:1, RXSFDD:1, RXPRD:1,
             % bits 15-8
        SPL2INIT:1, GPIOIRQ:1, RXPTO:1, RXOVRR:1, Reserved0:1, LDEERR:1, RXRFTO:1,
             RXRFSL:1, % bits 23-16
        ICRBP:1, HSRBP:1, AFFREJ:1, TXBERR:1, HPDWARN:1, RXSFDTO:1, CLCKPLL_LL:1,
            RFPLL_LL:1, % bits 31-24
        Reserved1:5, TXPUTE:1, RXPREJ:1, RXRSCS:1 % bits 39-32
    >>;
reg(decode, rx_finfo, Resp) ->
    <<
        RXPACC:12, RXPSR:2, RXPRFR:2, RNG:1, RXBR:2, RXNSPL:2, _:1, RXFLE:3,
            RXFLEN:7
    >> = reverse(Resp),
    #{
        rxpacc => RXPACC, rxpsr => RXPSR, rxprfr => RXPRFR, rng => RNG, rxbr =>
            RXBR, rxnspl => RXNSPL, rxfle => RXFLE, rxflen => RXFLEN
    };
reg(decode, rx_buffer, Resp) ->
    #{ rx_buffer => reverse(Resp)};
reg(decode, rx_fqual, Resp) ->
    <<
        CIR_PWR:16, PP_AMPL3:16, FP_AMPL2:16, STD_NOISE:16
    >> = Resp,
    #{
        cir_pwr => CIR_PWR, pp_ampl3 => PP_AMPL3, fp_ampl2 => FP_AMPL2, std_noise
            => STD_NOISE
    };
reg(decode, rx_ttcki, Resp) ->
    <<
      RXTTCKI:32
    >> = reverse(Resp),
    #{
        rxttcki => RXTTCKI
    };
reg(decode, rx_ttcko, Resp) ->
    <<
        _:1, RCPHASE:7, RSMPDEL:8, _:5, RXTOFS:19
    >> = reverse(Resp),
    #{
        rcphase => RCPHASE, rsmpdel => RSMPDEL, rxtofs => RXTOFS
    };
reg(decode, rx_time, Resp) ->
    <<
        RX_RAWST:40, FP_AMPL1:16, FP_INDEX:16, RX_STAMP:40
    >> = reverse(Resp),
    #{
        rx_rawst => RX_RAWST, fp_ampl1 => FP_AMPL1, fp_index => FP_INDEX, rx_stamp
             => RX_STAMP
    };
reg(decode, tx_time, Resp) ->
    <<
        TX_RAWST:40, TX_STAMP:40
```

```
    >> = reverse(Resp),
    #{
        tx_rawst => TX_RAWST, tx_stamp => TX_STAMP
    };
reg(decode, tx_antd, Resp) ->
    #{
        tx_antd => reverse(Resp)
    };
reg(encode, tx_antd, Val) ->
    #{
        tx_antd := TX_ANTD
    } = Val,
    reverse(<<
        TX_ANTD:16
    >>);
reg(decode, sys_state, Resp) ->
    <<
        _:8, _:4,  PMSC_STATE:4, _:3, RX_STATE:5, _:4, TX_STATE:4
    >> = reverse(Resp),
    #{
        pmsc_state => PMSC_STATE, rx_state => RX_STATE, tx_state => TX_STATE
    };
reg(decode, ack_resp_t, Resp) ->
    <<
        ACK_TIME:8, _:4, W4R_TIME:20
    >> = reverse(Resp),
    #{
        ack_tim => ACK_TIME, w4r_tim => W4R_TIME
    };
reg(encode, ack_resp_t, Val) ->
    #{
        ack_tim := ACK_TIME, w4r_tim := W4R_TIME
    } = Val,
    reverse(<<
        ACK_TIME:8, 2#0:4, W4R_TIME:20
    >>);
reg(decode, rx_sniff, Resp) ->
    <<
        Reserved0:16, SNIFF_OFFT:8, Reserved1:4, SNIFF_ONT:4
    >> = reverse(Resp),
    #{
        res0 => Reserved0,
        sniff_offt => SNIFF_OFFT,
        sniff_ont => SNIFF_ONT,
        res1 => Reserved1
    };
reg(encode, rx_sniff, Val) ->
    #{
        res0 := Reserved0,
        sniff_offt := SNIFF_OFFT,
        sniff_ont := SNIFF_ONT,
        res1 := Reserved1
    } = Val,
    reverse(<<
        Reserved0:16, SNIFF_OFFT:8, Reserved1:4, SNIFF_ONT:4
    >>);
% Smart transmit power control (cf. user manual p 104)
reg(decode, tx_power, Resp) ->
    <<
        BOOSTP125:8, BOOSTP250:8, BOOSTP500:8, BOOSTNORM:8
    >> = reverse(Resp),
    #{
```

```
        boostp125 => BOOSTP125, boostp250 => BOOSTP250, boostp500 => BOOSTP500,
            boostnorm => BOOSTNORM
    };
reg(encode, tx_power, Val) ->
    % Leave the possibility to the user to write the value as one
    case Val of
        #{ tx_power := ValToEncode } -> reverse(<<ValToEncode:32>>);
        #{ boostp125 := BOOSTP125, boostp250 := BOOSTP250, boostp500 := BOOSTP500,
             boostnorm := BOOSTNORM } ->reverse(<<BOOSTP125:8, BOOSTP250:8,
            BOOSTP500:8, BOOSTNORM:8>>)
    end;
reg(decode, chan_ctrl, Resp) ->
    <<
        RX_PCODE:5, TX_PCODE:5, RNSSFD:1, TNSSFD:1, RXPRF:2, DWSFD:1, Reserved0:9,
             RX_CHAN:4, TX_CHAN:4
    >> = reverse(Resp),
    #{
        rx_pcode => RX_PCODE, tx_pcode => TX_PCODE, rnssfd => RNSSFD, tnssfd =>
            TNSSFD, rxprf => RXPRF, dwsfd => DWSFD, res0 => Reserved0, rx_chan =>
            RX_CHAN, tx_chan => TX_CHAN
    };
reg(encode, chan_ctrl, Val) ->
    #{
        rx_pcode := RX_PCODE, tx_pcode := TX_PCODE, rnssfd := RNSSFD, tnssfd :=
            TNSSFD, rxprf := RXPRF, dwsfd := DWSFD, res0 := Reserved0, rx_chan :=
            RX_CHAN, tx_chan := TX_CHAN
    } = Val,
    reverse(<<
        RX_PCODE:5, TX_PCODE:5, RNSSFD:1, TNSSFD:1, RXPRF:2, DWSFD:1, Reserved0:9,
             RX_CHAN:4, TX_CHAN:4
    >>);
reg(encode, usr_sfd, Value) ->
    #{
      usr_sfd := USR_SFD
     } = Value,
    reverse(<<
        USR_SFD:(8*41)
    >>);
reg(decode, usr_sfd, Resp) ->
    <<
        USR_SFD:(8*41)
    >> = reverse(Resp),
    #{
      usr_sfd => USR_SFD
     };
% AGC_CTRL is a complex register with reserved bits that can't be written
reg(encode, agc_ctrl1, Val) ->
    #{
        res := Reserved, dis_am := DIS_AM
    } = Val,
    reverse(<<
        Reserved:15, DIS_AM:1
    >>);
reg(encode, agc_tune1, Val) ->
    #{
      agc_tune1 := AGC_TUNE1
     } = Val,
    reverse(<<
        AGC_TUNE1:16
    >>);
reg(encode, agc_tune2, Val) ->
    #{
```

```
        agc_tune2 := AGC_TUNE2
     } = Val,
    reverse(<<
        AGC_TUNE2:32
    >>);
reg(encode, agc_tune3, Val) ->
    #{
        agc_tune3 := AGC_TUNE3
     } = Val,
    reverse(<<
        AGC_TUNE3:16
    >>);
reg(decode, agc_ctrl, Resp) ->
    <<
        _:4, EDV2:9, EDG1:5, _:6, % AGC_STAT1 (RP => don't save reserved bits)
        _:80, % Reserved 4
        AGC_TUNE3:16, % AGC_TUNE3
        _:16, % Reserved 3
        AGC_TUNE2:32, % AGC_TUNE2
        _:48, % Reserved 2
        AGC_TUNE1:16, % AGC_TUNE1
        Reserved0:15, DIS_AM:1, % AGC_CTRL1 (RW => save reserved bits)
        _:16 % Reserved 1
    >> = reverse(Resp),
    #{
        agc_ctrl1 => #{res => Reserved0, dis_am => DIS_AM},
        agc_tune1 => AGC_TUNE1,
        agc_tune2 => AGC_TUNE2,
        agc_tune3 => AGC_TUNE3,
        agc_stat1 => #{edv2 => EDV2, edg1 => EDG1}
    };
reg(encode, ec_ctrl, Val) ->
    #{
       res := Reserved, ostrm := OSTRM, wait := WAIT, pllldt := PLLLDT, osrsm := OSRSM, ostsm := OSTSM
     } = Val,
    reverse(<<
        Reserved:20, OSTRM:1, WAIT:8, PLLLDT:1, OSRSM:1, OSTSM:1 % EC_CTRL
    >>);
reg(decode, ext_sync, Resp) ->
    <<
        _:26, OFFSET_EXT:6, % EC_GLOP
        RX_TS_EST:32, % EC_RXTC
        Reserved:20, OSTRM:1, WAIT:8, PLLLDT:1, OSRSM:1, OSTSM:1 % EC_CTRL
    >> = reverse(Resp),
    #{
        ec_ctrl => #{res => Reserved, ostrm => OSTRM, wait => WAIT, pllldt => PLLLDT, osrsm => OSRSM, ostsm => OSTSM},
        rx_ts_est => RX_TS_EST,
        ec_golp => #{offset_ext => OFFSET_EXT}
    };
% "The host system doesn't need to access the ACC_MEM in normal operation, however it may be of interest [...] for diagnostic purpose" (from DW1000 user manual)
reg(decode, acc_mem, Resp) ->
    #{
        acc_mem => reverse(Resp)
    };
reg(encode, gpio_mode, Val) ->
    #{
      msgp8 := MSGP8, msgp7 := MSGP7, msgp6 := MSGP6, msgp5 := MSGP5, msgp4 := MSGP4, msgp3 := MSGP3, msgp2 := MSGP2, msgp1 := MSGP1, msgp0 := MSGP0
     } = Val,
```

```
    reverse(<<
        2#0:8, MSGP8:2, MSGP7:2, MSGP6:2, MSGP5:2, MSGP4:2, MSGP3:2, MSGP2:2,
            MSGP1:2, MSGP0:2, 2#0:6 % GPIO_MODE
    >>);
reg(encode, gpio_dir, Val) ->
    #{
        gdm8 := GDM8, gdm7 := GDM7, gdm6 := GDM6, gdm5 := GDM5, gdm4 := GDM4, gdm3
             := GDM3, gdm2 := GDM2, gdm1 := GDM1, gdm0 := GDM0,
        gdp8 := GDP8, gdp7 := GDP7, gdp6 := GDP6, gdp5 := GDP5, gdp4 := GDP4, gdp3
             := GDP3, gdp2 := GDP2, gdp1 := GDP1, gdp0 := GDP0
     } = Val,
    reverse(<<
        2#0:11, GDM8:1, 2#0:3, GDP8:1, GDM7:1, GDM6:1, GDM5:1, GDM4:1, GDP7:1,
            GDP6:1, GDP5:1, GDP4:1, GDM3:1, GDM2:1, GDM1:1, GDM0:1, GDP3:1, GDP2
            :1, GDP1:1, GDP0:1 % GPIO2_DIR
    >>);
reg(encode, gpio_dout, Val) ->
    #{
        gom8 := GOM8, gom7 := GOM7, gom6 := GOM6, gom5 := GOM5, gom4 := GOM4, gom3
             := GOM3, gom2 := GOM2, gom1 := GOM1, gom0 := GOM0,
        gop8 := GOP8, gop7 := GOP7, gop6 := GOP6, gop5 := GOP5, gop4 := GOP4, gop3
             := GOP3, gop2 := GOP2, gop1 := GOP1, gop0 := GOP0
     } = Val,
    reverse(<<
        2#0:11, GOM8:1, 2#0:3, GOP8:1, GOM7:1, GOM6:1, GOM5:1, GOM4:1, GOP7:1,
            GOP6:1, GOP5:1, GOP4:1, GOM3:1, GOM2:1, GOM1:1, GOM0:1, GOP3:1, GOP2
            :1, GOP1:1, GOP0:1 % GPIO_DOUT
    >>);
reg(encode, gpio_irqe, Val) ->
    #{
        girqe8 := GIRQE8, girqe7 := GIRQE7, girqe6 := GIRQE6, girqe5 := GIRQE5,
            girqe4 := GIRQE4, girqe3 := GIRQE3, girqe2 := GIRQE2, girqe1 := GIRQE1
            , girqe0 := GIRQE0
     } = Val,
    reverse(<<
        2#0:23, GIRQE8:1, GIRQE7:1, GIRQE6:1, GIRQE5:1, GIRQE4:1, GIRQE3:1, GIRQE2
            :1, GIRQE1:1, GIRQE0:1 % GPIO_IRQE
    >>);
reg(encode, gpio_isen, Val) ->
    #{
        gisen8 := GISEN8, gisen7 := GISEN7, gisen6 := GISEN6, gisen5 := GISEN5,
            gisen4 := GISEN4, gisen3 := GISEN3, gisen2 := GISEN2, gisen1 := GISEN1
            , gisen0 := GISEN0
     } = Val,
    reverse(<<
        2#0:23, GISEN8:1, GISEN7:1, GISEN6:1, GISEN5:1, GISEN4:1, GISEN3:1, GISEN2
            :1, GISEN1:1, GISEN0:1 % GPIO_ISEN
    >>);
reg(encode, gpio_imod, Val) ->
    #{
        gimod8 := GIMOD8, gimod7 := GIMOD7, gimod6 := GIMOD6, gimod5 := GIMOD5,
            gimod4 := GIMOD4, gimod3 := GIMOD3, gimod2 := GIMOD2, gimod1 := GIMOD1
            , gimod0 := GIMOD0
     } = Val,
    reverse(<<
        2#0:23, GIMOD8:1, GIMOD7:1, GIMOD6:1, GIMOD5:1, GIMOD4:1, GIMOD3:1, GIMOD2
            :1, GIMOD1:1, GIMOD0:1 % GPIO_IMOD
    >>);
reg(encode, gpio_ibes, Val) ->
    #{
        gibes8 := GIBES8, gibes7 := GIBES7, gibes6 := GIBES6, gibes5 := GIBES5,
            gibes4 := GIBES4, gibes3 := GIBES3, gibes2 := GIBES2, gibes1 := GIBES1
```

```
            , gibes0 := GIBES0
     } = Val,
    reverse(<<
        2#0:23, GIBES8:1, GIBES7:1, GIBES6:1, GIBES5:1, GIBES4:1, GIBES3:1, GIBES2:1, GIBES1:1, GIBES0:1 % GPIO_IBES
    >>);
reg(encode, gpio_iclr, Val) ->
    #{
        giclr8 := GICLR8, giclr7 := GICLR7, giclr6 := GICLR6, giclr5 := GICLR5, giclr4 := GICLR4, giclr3 := GICLR3, giclr2 := GICLR2, giclr1 := GICLR1, giclr0 := GICLR0
     } = Val,
    reverse(<<
        2#0:23, GICLR8:1, GICLR7:1, GICLR6:1, GICLR5:1, GICLR4:1, GICLR3:1, GICLR2:1, GICLR1:1, GICLR0:1 % GPIO_ICLR
    >>);
reg(encode, gpio_idbe, Val) ->
    #{
        gidbe8 := GIDBE8, gidbe7 := GIDBE7, gidbe6 := GIDBE6, gidbe5 := GIDBE5, gidbe4 := GIDBE4, gidbe3 := GIDBE3, gidbe2 := GIDBE2, gidbe1 := GIDBE1, gidbe0 := GIDBE0
     } = Val,
    reverse(<<
        2#0:23, GIDBE8:1, GIDBE7:1, GIDBE6:1, GIDBE5:1, GIDBE4:1, GIDBE3:1, GIDBE2:1, GIDBE1:1, GIDBE0:1 % GPIO_IDBE
    >>);
reg(encode, gpio_raw, Val) ->
    #{
        grawp8 := GRAWP8, grawp7 := GRAWP7, grawp6 := GRAWP6, grawp5 := GRAWP5, grawp4 := GRAWP4, grawp3 := GRAWP3, grawp2 := GRAWP2, grawp1 := GRAWP1, grawp0 := GRAWP0
     } = Val,
    reverse(<<
        2#0:23, GRAWP8:1, GRAWP7:1, GRAWP6:1, GRAWP5:1, GRAWP4:1, GRAWP3:1, GRAWP2:1, GRAWP1:1, GRAWP0:1 % GPIO_RAW
    >>);
reg(decode, gpio_ctrl, Resp) ->
    <<
        _:23, GRAWP8:1, GRAWP7:1, GRAWP6:1, GRAWP5:1, GRAWP4:1, GRAWP3:1, GRAWP2:1, GRAWP1:1, GRAWP0:1, % GPIO_RAW
        _:23, GIDBE8:1, GIDBE7:1, GIDBE6:1, GIDBE5:1, GIDBE4:1, GIDBE3:1, GIDBE2:1, GIDBE1:1, GIDBE0:1, % GPIO_IDBE
        _:23, GICLR8:1, GICLR7:1, GICLR6:1, GICLR5:1, GICLR4:1, GICLR3:1, GICLR2:1, GICLR1:1, GICLR0:1, % GPIO_ICLR
        _:23, GIBES8:1, GIBES7:1, GIBES6:1, GIBES5:1, GIBES4:1, GIBES3:1, GIBES2:1, GIBES1:1, GIBES0:1, % GPIO_IBES
        _:23, GIMOD8:1, GIMOD7:1, GIMOD6:1, GIMOD5:1, GIMOD4:1, GIMOD3:1, GIMOD2:1, GIMOD1:1, GIMOD0:1, % GPIO_IMOD
        _:23, GISEN8:1, GISEN7:1, GISEN6:1, GISEN5:1, GISEN4:1, GISEN3:1, GISEN2:1, GISEN1:1, GISEN0:1, % GPIO_ISEN
        _:23, GIRQE8:1, GIRQE7:1, GIRQE6:1, GIRQE5:1, GIRQE4:1, GIRQE3:1, GIRQE2:1, GIRQE1:1, GIRQE0:1, % GPIO_IRQE
        _:11, GOM8:1, _:3, GOP8:1, GOM7:1, GOM6:1, GOM5:1, GOM4:1, GOP7:1, GOP6:1, GOP5:1, GOP4:1, GOM3:1, GOM2:1, GOM1:1, GOM0:1, GOP3:1, GOP2:1, GOP1:1, GOP0:1, % GPIO_DOUT
        _:11, GDM8:1, _:3, GDP8:1, GDM7:1, GDM6:1, GDM5:1, GDM4:1, GDP7:1, GDP6:1, GDP5:1, GDP4:1, GDM3:1, GDM2:1, GDM1:1, GDM0:1, GDP3:1, GDP2:1, GDP1:1, GDP0:1, % GPIO_DIR
        _:32, % Reserved
        _:8, MSGP8:2, MSGP7:2, MSGP6:2, MSGP5:2, MSGP4:2, MSGP3:2, MSGP2:2, MSGP1:2, MSGP0:2, _:6 % GPIO_MODE
    >> = reverse(Resp),
```

```
    #{
        gpio_mode => #{msgp8 => MSGP8, msgp7 => MSGP7, msgp6 => MSGP6, msgp5 => MSGP5, msgp4 => MSGP4, msgp3 => MSGP3, msgp2 => MSGP2, msgp1 => MSGP1, msgp0 => MSGP0},
        gpio_dir => #{gdm8 => GDM8, gdm7 => GDM7, gdm6 => GDM6, gdm5 => GDM5, gdm4 => GDM4, gdm3 => GDM3, gdm2 => GDM2, gdm1 => GDM1, gdm0 => GDM0,
                      gdp8 => GDP8, gdp7 => GDP7, gdp6 => GDP6, gdp5 => GDP5, gdp4 => GDP4, gdp3 => GDP3, gdp2 => GDP2, gdp1 => GDP1, gdp0 => GDP0},
        gpio_dout => #{gom8 => GOM8, gom7 => GOM7, gom6 => GOM6, gom5 => GOM5, gom4 => GOM4, gom3 => GOM3, gom2 => GOM2, gom1 => GOM1, gom0 => GOM0,
                       gop8 => GOP8, gop7 => GOP7, gop6 => GOP6, gop5 => GOP5, gop4 => GOP4, gop3 => GOP3, gop2 => GOP2, gop1 => GOP1, gop0 => GOP0},
        gpio_irqe => #{girqe8 => GIRQE8, girqe7 => GIRQE7, girqe6 => GIRQE6, girqe5 => GIRQE5, girqe4 => GIRQE4, girqe3 => GIRQE3, girqe2 => GIRQE2, girqe1 => GIRQE1, girqe0 => GIRQE0},
        gpio_isen => #{gisen8 => GISEN8, gisen7 => GISEN7, gisen6 => GISEN6, gisen5 => GISEN5, gisen4 => GISEN4, gisen3 => GISEN3, gisen2 => GISEN2, gisen1 => GISEN1, gisen0 => GISEN0},
        gpio_imod => #{gimod8 => GIMOD8, gimod7 => GIMOD7, gimod6 => GIMOD6, gimod5 => GIMOD5, gimod4 => GIMOD4, gimod3 => GIMOD3, gimod2 => GIMOD2, gimod1 => GIMOD1, gimod0 => GIMOD0},
        gpio_ibes => #{gibes8 => GIBES8, gibes7 => GIBES7, gibes6 => GIBES6, gibes5 => GIBES5, gibes4 => GIBES4, gibes3 => GIBES3, gibes2 => GIBES2, gibes1 => GIBES1, gibes0 => GIBES0},
        gpio_iclr => #{giclr8 => GICLR8, giclr7 => GICLR7, giclr6 => GICLR6, giclr5 => GICLR5, giclr4 => GICLR4, giclr3 => GICLR3, giclr2 => GICLR2, giclr1 => GICLR1, giclr0 => GICLR0},
        gpio_idbe => #{gidbe8 => GIDBE8, gidbe7 => GIDBE7, gidbe6 => GIDBE6, gidbe5 => GIDBE5, gidbe4 => GIDBE4, gidbe3 => GIDBE3, gidbe2 => GIDBE2, gidbe1 => GIDBE1, gidbe0 => GIDBE0},
        gpio_raw => #{grawp8 => GRAWP8, grawp7 => GRAWP7, grawp6 => GRAWP6, grawp5 => GRAWP5, grawp4 => GRAWP4, grawp3 => GRAWP3, grawp2 => GRAWP2, grawp1 => GRAWP1, grawp0 => GRAWP0}
    };
reg(encode, drx_tune0b, Val) ->
    #{
        drx_tune0b := DRX_TUNE0b
     } = Val,
    reverse(<<
        DRX_TUNE0b:16
    >>);
reg(encode, drx_tune1a, Val) ->
    #{
        drx_tune1a := DRX_TUNE1a
     } = Val,
    reverse(<<
        DRX_TUNE1a:16
    >>);
reg(encode, drx_tune1b, Val) ->
    #{
        drx_tune1b := DRX_TUNE1b
     } = Val,
    reverse(<<
        DRX_TUNE1b:16
    >>);
reg(encode, drx_tune2, Val) ->
    #{
        drx_tune2 := DRX_TUNE2
     } = Val,
    reverse(<<
```

```
        DRX_TUNE2:32
    >>);
reg(encode, drx_sfdtoc, Val) ->
    #{
        drx_sfdtoc := DRX_SFDTOC
     } = Val,
    reverse(<<
        DRX_SFDTOC:16
    >>);
reg(encode, drx_pretoc, Val) ->
    #{
        drx_pretoc := DRX_PRETOC
     } = Val,
    reverse(<<
        DRX_PRETOC:16
    >>);
reg(encode, drx_tune4h, Val) ->
    #{
        drx_tune4h := DRX_TUNE4H
     } = Val,
    reverse(<<
        DRX_TUNE4H:16
    >>);
reg(decode, drx_conf, Resp) ->
    <<
        RXPACC_NOSAT:8, % present in the user manual but not in the driver code in C
        % _:8, % Placeholder for the remaining 8 bits
        DRX_CAR_INT:24,
        DRX_TUNE4H:16,
        DRX_PRETOC:16,
        _:16,
        DRX_SFDTOC:16,
        _:160,
        DRX_TUNE2:32,
        DRX_TUNE1b:16,
        DRX_TUNE1a:16,
        DRX_TUNE0b:16,
        _:16
    >> = reverse(Resp),
    #{
        drx_tune0b => DRX_TUNE0b,
        drx_tune1a => DRX_TUNE1a,
        drx_tune1b => DRX_TUNE1b,
        drx_tune2 => DRX_TUNE2,
        drx_tune4h => DRX_TUNE4H,
        drx_car_int => DRX_CAR_INT,
        drx_sfdtoc => DRX_SFDTOC,
        drx_pretoc => DRX_PRETOC,
        rxpacc_nosat => RXPACC_NOSAT
    };
reg(encode, rf_conf, Val) ->
    #{
        txrxsw := TXRXSW, ldofen := LDOFEN, pllfen := PLLFEN, txfen := TXFEN
     } = Val,
    reverse(<<
        2#0:9, TXRXSW:2, LDOFEN:5, PLLFEN:3, TXFEN:5, 2#0:8 % RF_CONF
    >>);
reg(encode, rf_rxctrlh, Val) ->
    #{
        rf_rxctrlh := RF_RXCTRLH
     } = Val,
```

```
    reverse(<<
        RF_RXCTRLH:8 % RF_RXCTRLH
    >>);
% user manual gives fields but encoding should be done as one following table 38
reg(encode, rf_txctrl, Val) ->
    #{
      rf_txctrl := RF_TXCTRL
     } = Val,
    reverse(<<
        RF_TXCTRL:32
    >>);
reg(encode, ldotune, Val) ->
    #{
      ldotune := LDOTUNE
     } = Val,
    reverse(<<
        LDOTUNE:40
    >>);
reg(decode, rf_conf, Resp) ->
    <<
        _:40, % Placeholder for the remaining 40 bits
        LDOTUNE:40, % LDOTUNE
        _:28, RFPLLLOCK:1, CPLLHIGH:1, CPLLLOW:1, CPLLLOCK:1, % RF_STATUS
        _:128, _:96, % Reserved 2 - On user manual 16 bytes but offset gives 28 bytes (16 bytes (128 bits) + 12 bytes (96 bits))
        RF_TXCTRL:32, % cf. encode function: Reserved:20, TXMQ:3, TXMTUNE:4, _:5 - RF_TXCTRL
        RF_RXCTRLH:8, % RF_RXCTRLH
        _:56, % Reserved 1
        _:9, TXRXSW:2, LDOFEN:5, PLLFEN:3, TXFEN:5, _:8 % RF_CONF
    >> = reverse(Resp),
    #{
        ldotune => LDOTUNE,
        rf_status => #{rfplllock => RFPLLLOCK, cplllow => CPLLLOW, cpllhigh => CPLLHIGH, cplllock => CPLLLOCK},
        rf_txctrl => RF_TXCTRL,
        rf_rxctrlh => RF_RXCTRLH,
        rf_conf => #{txrxsw => TXRXSW, ldofen => LDOFEN, pllfen => PLLFEN, txfen => TXFEN}
    };
reg(encode, tc_sarc, Val) ->
    #{
        sar_ctrl := SAR_CTRL
     } = Val,
    reverse(<<
      2#0:15, SAR_CTRL:1
    >>);
reg(encode, tc_pg_ctrl, Val) ->
    #{
        pg_tmeas := PG_TMEAS, res := Reserved, pg_start := PG_START
     } = Val,
    reverse(<<
        2#0:2, PG_TMEAS:4, Reserved:1, PG_START:1
    >>);
reg(encode, tc_pgdelay, Val) ->
    #{
        tc_pgdelay := TC_PGDELAY
     } = Val,
    reverse(<<
        TC_PGDELAY:8
    >>);
reg(encode, tc_pgtest, Val) ->
```

```
    #{
        tc_pgtest := TC_PGTEST
     } = Val,
    reverse(<<
        TC_PGTEST:8
    >>);
reg(decode, tx_cal, Resp) ->
    <<
        TC_PGTEST:8, % TC_PGTEST
        TC_PGDELAY:8, % TC_PGDELAY
        _:4, DELAY_CNT:12, % TC_PG_STATUS
        _:2, PG_TMEAS:4, Reserved0:1, PG_START:1, % TC_PG_CTRL
        SAR_WTEMP:8, SAR_WVBAT:8, % TC_SARW
        _:8, SAR_LTEMP:8, SAR_LVBAT:8, % TC_SARL
        _:8, % Place holder to fill the gap between the offsets
        _:15, SAR_CTRL:1 % TC_SARC
    >> = reverse(Resp),
    #{
        tc_pgtest => TC_PGTEST,
        tc_pgdelay => TC_PGDELAY,
        tc_pg_status => #{delay_cnt => DELAY_CNT},
        tc_pg_ctrl => #{pg_tmeas => PG_TMEAS, res => Reserved0, pg_start =>
            PG_START},
        tc_sarw => #{sar_wtemp => SAR_WTEMP, sar_wvbat => SAR_WVBAT},
        tc_sarl => #{sar_ltemp => SAR_LTEMP, sar_lvbat => SAR_LVBAT},
        tc_sarc => #{sar_ctrl => SAR_CTRL}
    };
reg(encode, fs_pllcfg, Val) ->
    #{
        fs_pllcfg := FS_PLLCFG
     } = Val,
    reverse(<<
        FS_PLLCFG:32
    >>);
reg(encode, fs_plltune, Val) ->
    #{
        fs_plltune := FS_PLLTUNE
     } = Val,
    reverse(<<
        FS_PLLTUNE:8
    >>);
reg(encode, fs_xtalt, Val) ->
    #{
      res := Reserved, xtalt := XTALT
     } = Val,
    reverse(<<
        Reserved:3, XTALT:5
    >>);
reg(decode, fs_ctrl, Resp) ->
    <<
        _:48, % Reserved 3
        Reserved:3, XTALT:5, % FS_XTALT
        _:16, % Reserved 2
        FS_PLLTUNE:8, % FS_PLLTUNE
        FS_PLLCFG:32, % FS_PLLCFG
        _:56 % Reserved 1
    >> = reverse(Resp),
    #{
        fs_xtalt => #{res => Reserved, xtalt => XTALT},
        fs_plltune => FS_PLLTUNE,
        fs_pllcfg => FS_PLLCFG
    };
```

```
reg(encode, aon_wcfg, Val) ->
    #{
        onw_lld := ONW_LLD, onw_llde := ONW_LLDE, pres_slee := PRES_SLEE, own_l64 := OWN_L64, own_ldc := OWN_LDC, own_leui := OWN_LEUI, own_rx := OWN_RX, own_rad := OWN_RAD
     } = Val,
    reverse(<<
        2#0:3, ONW_LLD:1, ONW_LLDE:1, 2#0:2, PRES_SLEE:1, OWN_L64:1, OWN_LDC:1, 2#0:2, OWN_LEUI:1, 2#0:1, OWN_RX:1, OWN_RAD:1 % AON_WCFG
    >>);
reg(encode, aon_ctrl, Val) ->
    #{
        dca_enab := DCA_ENAB, dca_read := DCA_READ, upl_cfg := UPL_CFG, save := SAVE, restore := RESTORE
     } = Val,
    reverse(<<
        DCA_ENAB:1, 2#0:3, DCA_READ:1, UPL_CFG:1, SAVE:1, RESTORE:1 % AON_CTRL
    >>);
reg(encode, aon_rdat, Val) ->
    #{
        aon_rdat := AON_RDAT
     } = Val,
    reverse(<<
        AON_RDAT:8 % AON_RDAT
    >>);
reg(encode, aon_addr, Val) ->
    #{
        aon_addr := AON_ADDR
     } = Val,
    reverse(<<
        AON_ADDR:8 % AON_ADDR
    >>);
reg(encode, aon_cfg0, Val) ->
    #{
        sleep_tim := SLEEP_TIM, lpclkdiva := LPCLKDIVA, lpdiv_en := LPDIV_EN, wake_cnt := WAKE_CNT, wake_spi := WAKE_SPI, wake_pin := WAKE_PIN, sleep_en := SLEEP_EN
     } = Val,
    reverse(<<
        SLEEP_TIM:16, LPCLKDIVA:11, LPDIV_EN:1, WAKE_CNT:1, WAKE_SPI:1, WAKE_PIN:1, SLEEP_EN:1 % AON_CFG0
    >>);
reg(encode, aon_cfg1, Val) ->
    #{
        res := Reserved, lposc_c := LPOSC_C, smxx := SMXX, sleep_ce := SLEEP_CE
     } = Val,
    reverse(<<
        Reserved:13, LPOSC_C:1, SMXX:1, SLEEP_CE:1 % AON_CFG1
    >>);
reg(decode, aon, Resp) ->
    <<
        Reserved:13, LPOSC_C:1, SMXX:1, SLEEP_CE:1, % AON_CFG1
        SLEEP_TIM:16, LPCLKDIVA:11, LPDIV_EN:1, WAKE_CNT:1, WAKE_SPI:1, WAKE_PIN:1, SLEEP_EN:1, % AON_CFG0
        _:8, % Reserved 1
        AON_ADDR:8, % AON_ADDR
        AON_RDAT:8, % AON_RDAT
        DCA_ENAB:1, _:3, DCA_READ:1, UPL_CFG:1, SAVE:1, RESTORE:1, % AON_CTRL
        _:3, ONW_LLD:1, ONW_LLDE:1, _:2, PRES_SLEE:1, OWN_L64:1, OWN_LDC:1, _:2, OWN_LEUI:1, _:1, OWN_RX:1, OWN_RAD:1 % AON_WCFG
    >> = reverse(Resp),
    #{
```

```
        aon_cfg1 => #{res => Reserved, lposc_c => LPOSC_C, smxx => SMXX, sleep_ce
            => SLEEP_CE},
        aon_cfg0 => #{sleep_tim => SLEEP_TIM, lpclkdiva => LPCLKDIVA, lpdiv_en =>
            LPDIV_EN, wake_cnt => WAKE_CNT, wake_spi => WAKE_SPI, wake_pin =>
            WAKE_PIN, sleep_en => SLEEP_EN},
        aon_addr => AON_ADDR,
        aon_rdat => AON_RDAT,
        aon_ctrl => #{dca_enab => DCA_ENAB, dca_read => DCA_READ, upl_cfg =>
            UPL_CFG, save => SAVE, restore => RESTORE},
        aon_wcfg => #{onw_lld => ONW_LLD, onw_llde => ONW_LLDE, pres_slee =>
            PRES_SLEE, own_l64 => OWN_L64, own_ldc => OWN_LDC, own_leui =>
            OWN_LEUI, own_rx => OWN_RX, own_rad => OWN_RAD}
    };
reg(encode, otp_wdat, Val) ->
    #{
        otp_wdat := OTP_WDAT
     } = Val,
    reverse(<<
        OTP_WDAT:32 % OTP_WDAT
    >>);
reg(encode, otp_addr, Val) ->
    #{
        otpaddr := OTP_ADDR, res := Reserved
     } = Val,
    reverse(<<
        Reserved:5, OTP_ADDR:11 % OTP_ADDR
    >>);
reg(encode, otp_ctrl, Val) ->
    #{
        ldeload := LDELOAD, res1 := Reserved1, otpmr := OTPMR, otpprog := OTPPROG,
             res2 := Reserved2, otpmrwr := OTPMRWR, res3 := Reserved3, otpread :=
            OTPREAD, otp_rden := OTPRDEN
     } = Val,
    reverse(<<
        LDELOAD:1, Reserved1:4, OTPMR:4, OTPPROG:1, Reserved2:2, OTPMRWR:1,
            Reserved3:1, OTPREAD:1, OTPRDEN:1 % OTP_CTRL
    >>);
reg(encode, otp_stat, Val) ->
    #{
        res := Reserved, otp_vpok := OTP_VPOK, otpprgd := OTPPRGD
     } = Val,
    reverse(<<
        Reserved:14, OTP_VPOK:1, OTPPRGD:1 % OTP_STAT
    >>);
reg(encode, otp_rdat, Val) ->
    #{
        otp_rdat := OTP_RDAT
     } = Val,
    reverse(<<
        OTP_RDAT:32 % OTP_RDAT
    >>);
reg(encode, opt_srdat, Val) ->
    #{
        otp_srdat := OTP_SRDAT
     } = Val,
    reverse(<<
        OTP_SRDAT:32 % OTP_SRDAT
    >>);
reg(encode, otp_sf, Val) ->
    #{
        res1 := Reserved1, ops_sel := OPS_SEL, res2 := Reserved2, ldo_kick :=
            LDO_KICK, ops_kick := OPS_KICK
```

```
     } = Val,
    reverse(<<
        Reserved1:2, OPS_SEL:1, Reserved2:3, LDO_KICK:1, OPS_KICK:1 % OTP_SF
    >>);
reg(decode, otp_if, Resp) ->
    <<
        Reserved5:2, OPS_SEL:1, Reserved6:3, LDO_KICK:1, OPS_KICK:1, % OTP_SF
        OTP_SRDAT:32, % OTP_SRDAT
        OTP_RDAT:32, % OTP_RDAT
        Reserved4:14, OTP_VPOK:1, OTPPRGD:1, % OTP_STAT
        LDELOAD:1, Reserved1:4, OTPMR:4, OTPPROG:1, Reserved2:2, OTPMRWR:1,
            Reserved3:1, OTPREAD:1, OTPRDEN:1, % OTP_CTRL
        Reserved0:5, OTP_ADDR:11, % OTP_ADDR
        OTP_WDAT:32 % OTP_WDAT
    >> = reverse(Resp),
    #{
        otp_sf => #{res1 => Reserved5, ops_sel => OPS_SEL, res2 => Reserved6,
            ldo_kick => LDO_KICK, ops_kick => OPS_KICK},
        otp_srdat => OTP_SRDAT,
        otp_rdat => OTP_RDAT,
        otp_stat => #{res => Reserved4, otp_vpok => OTP_VPOK, otpprgd => OTPPRGD},
        otp_ctrl => #{ldeload => LDELOAD, res1 => Reserved1, otpmr => OTPMR,
            otpprog => OTPPROG, res2 => Reserved2, otpmrwr => OTPMRWR, res3 =>
            Reserved3, otpread => OTPREAD, otp_rden => OTPRDEN},
        otp_addr => #{otpaddr => OTP_ADDR, res => Reserved0},
        otp_wdat => OTP_WDAT
    };
reg(decode, lde_thresh, Resp) ->
    <<
      LDE_THRESH:16
    >> = reverse(Resp),
    #{
      lde_thresh => LDE_THRESH
    };
reg(encode, lde_cfg1, Val) ->
    #{
      pmult := PMULT, ntm := NTM
     } = Val,
    reverse(<<
        PMULT:3, NTM:5
    >>);
reg(decode, lde_cfg1, Resp) ->
    <<
      PMULT:3, NTM:5
    >> = reverse(Resp),
    #{
      lde_cfg1 => #{pmult => PMULT, ntm => NTM}
    };
reg(decode, lde_ppindx, Resp) ->
    <<
      LDE_PPINDX:16
    >> = reverse(Resp),
    #{
      lde_ppindx => LDE_PPINDX
    };
reg(decode, lde_ppampl, Resp) ->
    <<
      LDE_PPAMPL:16
    >> = reverse(Resp),
    #{
      lde_ppampl => LDE_PPAMPL
    };
```

```
reg(encode, lde_rxantd, Val) ->
    #{
      lde_rxantd := LDE_RXANTD
     } = Val,
    reverse(<<
        LDE_RXANTD:16
    >>);
reg(decode, lde_rxantd, Resp) ->
    <<
      LDE_RXANTD:16
    >> = reverse(Resp),
    #{
      lde_rxantd => LDE_RXANTD
    };
reg(encode, lde_cfg2, Val) ->
    #{
        lde_cfg2 := LDE_CFG2
     } = Val,
    reverse(<<
        LDE_CFG2:16
    >>);
reg(decode, lde_cfg2, Resp) ->
    <<
      LDE_CFG2:16
    >> = reverse(Resp),
    #{
      lde_cfg2 => LDE_CFG2
    };
reg(encode, lde_repc, Val) ->
    #{
        lde_repc := LDE_REPC
     } = Val,
    reverse(<<
        LDE_REPC:16
    >>);
reg(decode, lde_repc, Resp) ->
    <<
      LDE_REPC:16
    >> = reverse(Resp),
    #{
      lde_repc => LDE_REPC
    };
reg(encode, evc_ctrl, Val) ->
    #{
        evc_clr := EVC_CLR, evc_en := EVC_EN
     } = Val,
    reverse(<<
        2#0:30, EVC_CLR:1, EVC_EN:1 % EVC_CTRL
    >>);
reg(encode, diag_tmc, Val) ->
    #{
        tx_pstm := TX_PSTM
     } = Val,
    reverse(<<
        2#0:11, TX_PSTM:1, 2#0:4 % DIAG_TMC
    >>);
reg(decode, dig_diag, Resp) ->
    <<
        _:11, TX_PSTM:1, _:4, % DIAG_TMC
        _:64, % Reserved 1
        _:4, EVC_TPW:12, % EVC_TPW
        _:4, EVC_HPW:12, % EVC_HPW
```

```
        _:4, EVC_TXFS:12, % EVC_TXFS
        _:4, EVC_FWTO:12, % EVC_FWTO
        _:4, EVC_PTO:12, % EVC_PTO
        _:4, EVC_STO:12, % EVC_STO
        _:4, ECV_OVR:12, % EVC_OVR
        _:4, EVC_FFR:12, % EVC_FFR
        _:4, EVC_FCE:12, % EVC_FCE
        _:4, EVC_FCG:12, % EVC_FCG
        _:4, EVC_RSE:12, % EVC_RSE
        _:4, EVC_PHE:12, % EVC_PHE
        _:30, EVC_CLR:1, EVC_EN:1 % EVC_CTRL
    >> = reverse(Resp),
    #{
        diag_tmc => #{tx_pstm => TX_PSTM},
        evc_tpw => EVC_TPW,
        evc_hpw => EVC_HPW,
        evc_txfs => EVC_TXFS,
        evc_fwto => EVC_FWTO,
        evc_pto => EVC_PTO,
        evc_sto => EVC_STO,
        evc_ovr => ECV_OVR,
        evc_ffr => EVC_FFR,
        evc_fce => EVC_FCE,
        evc_fcg => EVC_FCG,
        evc_rse => EVC_RSE,
        evc_phe => EVC_PHE,
        evc_ctrl => #{evc_clr => EVC_CLR, evc_en => EVC_EN}
    };
reg(encode, pmsc_ctrl0, Val) ->
    #{
        softreset := SOFTRESET, pll2_seq_en := PLL2_SEQ_EN, khzclken := KHZCLKEN,
            gpdrn := GPDRN, gpdce := GPDCE,
        gprn := GPRN, gpce := GPCE, amce := AMCE, adcce := ADCCE, otp := OTP, res8
             := Res8, res7 := Res7, face := FACE, txclks := TXCLKS, rxclks :=
            RXCLKS, sysclks := SYSCLKS % Here we need res8 for the initial config
            of the DW1000. We need to write it
     } = Val,
    reverse(<<
        SOFTRESET:4, 2#000:3, PLL2_SEQ_EN:1, KHZCLKEN:1, 2#011:3, GPDRN:1, GPDCE
            :1, GPRN:1, GPCE:1, AMCE:1, 2#0000:4, ADCCE:1, OTP:1, Res8:1, Res7:1,
            FACE:1, TXCLKS:2, RXCLKS:2, SYSCLKS:2 % PMSC_CTRL0
    >>);
reg(encode, pmsc_ctrl1, Val) ->
    #{
        khzclkdiv := KHZCLKDIV, lderune := LDERUNE, pllsyn := PLLSYN, snozr :=
            SNOZR, snoze := SNOZE, arxslp := ARXSLP, atxslp := ATXSLP, pktseq :=
            PKTSEQ, arx2init := ARX2INIT
     } = Val,
    reverse(<<
        KHZCLKDIV:6, 2#01000000:8, LDERUNE:1, 2#0:1, PLLSYN:1, SNOZR:1, SNOZE:1,
            ARXSLP:1, ATXSLP:1, PKTSEQ:8, 2#0:1, ARX2INIT:1, 2#0:1 % PMSC_CTRL1
    >>);
reg(encode, pmsc_snozt, Val) ->
    #{
        snoz_tim := SNOZ_TIM
     } = Val,
    reverse(<<
        SNOZ_TIM:8 % PMSC_SNOZT
    >>);
reg(encode, pmsc_txfseq, Val) ->
    #{
        txfineseq := TXFINESEQ
```

```
    } = Val,
    reverse(<<
        TXFINESEQ:16 % PMSC_TXFINESEQ
    >>);
reg(encode, pmsc_ledc, Val) ->
    #{
        res31 := RES31, blnknow := BLNKNOW, res15 := RES15, blnken := BLNKEN, blink_tim := BLINK_TIM
    } = Val,
    reverse(<<
        RES31:12, BLNKNOW:4, RES15:7, BLNKEN:1, BLINK_TIM:8 % PMSC_LEDC
    >>);
% mapping pmsc ctrl0 from: https://forum.qorvo.com/t/pmsc-ctrl0-bits8-15/746/3
reg(decode, pmsc, Resp) ->
    % User manual says: reserved bits should be preserved at their reset value => can hardcode their values ? Safe to do that ?
    <<
        Res31:12, BLNKNOW:4, Res15:7, BLNKEN:1, BLINK_TIM:8, % PMSC_LEDC
        TXFINESEQ:16, % PMSC_TXFINESEQ
        _:(25*8), % Reserved 2
        SNOZ_TIM:8, % PMSC_SNOZT
        _:32, % Reserved 1
        KHZCLKDIV:6, _:8, LDERUNE:1, _:1, PLLSYN:1, SNOZR:1, SNOZE:1, ARXSLP:1, ATXSLP:1, PKTSEQ:8, _:1, ARX2INIT:1, _:1, % PMSC_CTRL1
        SOFTRESET:4, _:3, PLL2_SEQ_EN:1, KHZCLKEN:1, _:3, GPDRN:1, GPDCE:1, GPRN:1, GPCE:1, AMCE:1, _:4, ADCCE:1, OTP:1, Res8:1, Res7:1, FACE:1, TXCLKS:2, RXCLKS:2, SYSCLKS:2 % PMSC_CTRL0
    >> = reverse(Resp),
    #{
        pmsc_ledc => #{res31 => Res31, blnknow => BLNKNOW, res15 => Res15, blnken => BLNKEN, blink_tim => BLINK_TIM},
        pmsc_txfseq => #{txfineseq => TXFINESEQ},
        pmsc_snozt => #{snoz_tim => SNOZ_TIM},
        pmsc_ctrl1 => #{khzclkdiv => KHZCLKDIV, lderune => LDERUNE, pllsyn => PLLSYN, snozr => SNOZR, snoze => SNOZE, arxslp => ARXSLP, atxslp => ATXSLP, pktseq => PKTSEQ, arx2init => ARX2INIT},
        pmsc_ctrl0 => #{softreset => SOFTRESET, pll2_seq_en => PLL2_SEQ_EN, khzclken => KHZCLKEN, gpdrn => GPDRN, gpdce => GPDCE, gprn => GPRN, gpce => GPCE, amce => AMCE, adcce => ADCCE, otp => OTP, res8 => Res8, res7 => Res7, face => FACE, txclks => TXCLKS, rxclks => RXCLKS, sysclks => SYSCLKS}
    };
reg(decode, RegFile, Resp) -> error({unknown_regfile_to_decode, RegFile, Resp});
reg(encode, RegFile, Resp) -> error({unknown_regfile_to_encode, RegFile, Resp}).

rw(read) -> 0;
rw(write) -> 1.

% Mapping of the different register IDs to their hexadecimal value
regFile(dev_id) -> 16#00;
regFile(eui) -> 16#01;
% 0x02 is reserved
regFile(panadr) -> 16#03;
regFile(sys_cfg) -> 16#04;
% 0x05 is reserved
regFile(sys_time) -> 16#06;
% 0x07 is reserved
regFile(tx_fctrl) -> 16#08;
regFile(tx_buffer) -> 16#09;
regFile(dx_time) -> 16#0A;
% 0x0B is reserved
regFile(rx_fwto) -> 16#0C;
```

```
regFile(sys_ctrl) -> 16#0D;
regFile(sys_mask) -> 16#0E;
regFile(sys_status) -> 16#0F;
regFile(rx_finfo) -> 16#10;
regFile(rx_buffer) -> 16#11;
regFile(rx_fqual) -> 16#12;
regFile(rx_ttcki) -> 16#13;
regFile(rx_ttcko) -> 16#14;
regFile(rx_time) -> 16#15;
% 0x16 is reserved
regFile(tx_time) -> 16#17;
regFile(tx_antd) -> 16#18;
regFile(sys_state) -> 16#19;
regFile(ack_resp_t) -> 16#1A;
% 0x1B is reserved
% 0x1C is reserved
regFile(rx_sniff) -> 16#1D;
regFile(tx_power) -> 16#1E;
regFile(chan_ctrl) -> 16#1F;
% 0x20 is reserved
regFile(usr_sfd) -> 16#21;
% 0x22 is reserved
regFile(agc_ctrl) -> 16#23;
regFile(ext_sync) -> 16#24;
regFile(acc_mem) -> 16#25;
regFile(gpio_ctrl) -> 16#26;
regFile(drx_conf) -> 16#27;
regFile(rf_conf) -> 16#28;
% 0x29 is reserved
regFile(tx_cal) -> 16#2A;
regFile(fs_ctrl) -> 16#2B;
regFile(aon) -> 16#2C;
regFile(otp_if) -> 16#2D;
regFile(lde_ctrl) -> regFile(lde_if); % No size ?
regFile(lde_if) -> 16#2E;
regFile(dig_diag) -> 16#2F;
% 0x30 - 0x35 are reserved
regFile(pmsc) -> 16#36;
% 0x37 - 0x3F are reserved
regFile(RegId) -> error({wrong_register_ID, RegId}).

% Only the writtable subregisters in SRW register files are present here
% AGC_CTRL
subReg(agc_ctrl1) -> 16#02;
subReg(agc_tune1) -> 16#04;
subReg(agc_tune2) -> 16#0C;
subReg(agc_tune3) -> 16#12;
subReg(agc_stat1) -> 16#1E;
subReg(ec_ctrl) -> 16#00;
subReg(gpio_mode) -> 16#00;
subReg(gpio_dir) -> 16#08;
subReg(gpio_dout) -> 16#0C;
subReg(gpio_irqe) -> 16#10;
subReg(gpio_isen) -> 16#14;
subReg(gpio_imode) -> 16#18;
subReg(gpio_ibes) -> 16#1C;
subReg(gpio_iclr) -> 16#20;
subReg(gpio_idbe) -> 16#24;
subReg(gpio_raw) -> 16#28;
subReg(drx_tune0b) -> 16#02;
subReg(drx_tune1a) -> 16#04;
subReg(drx_tune1b) -> 16#06;
```

```
subReg(drx_tune2) -> 16#08;
subReg(drx_sfdtoc) -> 16#20;
subReg(drx_pretoc) -> 16#24;
subReg(drx_tune4h) -> 16#26;
subReg(rf_conf) -> 16#00;
subReg(rf_rxctrlh) -> 16#0B;
subReg(rf_txctrl) -> 16#0C;
subReg(ldotune) -> 16#30;
subReg(tc_sarc) -> 16#00;
subReg(tc_pg_ctrl) -> 16#08;
subReg(tc_pgdelay) -> 16#0B;
subReg(tc_pgtest) -> 16#0C;
subReg(fs_pllcfg) -> 16#07;
subReg(fs_plltune) -> 16#0B;
subReg(fs_xtalt) -> 16#0E;
subReg(aon_wcfg) -> 16#00;
subReg(aon_ctrl) -> 16#02;
subReg(aon_rdat) -> 16#03;
subReg(aon_addr) -> 16#04;
subReg(aon_cfg0) -> 16#06;
subReg(aon_cfg1) -> 16#0A;
subReg(otp_wdat) -> 16#00;
subReg(otp_addr) -> 16#04;
subReg(otp_ctrl) -> 16#06;
subReg(otp_stat) -> 16#08;
subReg(otp_rdat) -> 16#0A;
subReg(otp_srdat) -> 16#0E;
subReg(otp_sf) -> 16#12;
subReg(lde_thresh) -> 16#00;
subReg(lde_cfg1) -> 16#806;
subReg(lde_ppindx) -> 16#1000;
subReg(lde_ppampl) -> 16#1002;
subReg(lde_rxantd) -> 16#1804;
subReg(lde_cfg2) -> 16#1806;
subReg(lde_repc) -> 16#2804;
subReg(evc_ctrl) -> 16#00;
subReg(diag_tmc) -> 16#24;
subReg(pmsc_ctrl0) -> 16#00;
subReg(pmsc_ctrl1) -> 16#04;
subReg(pmsc_snozt) -> 16#0C;
subReg(pmsc_txfseq) -> 16#26;
subReg(pmsc_ledc) -> 16#28.


% Mapping of the size in bytes of the different register IDs
regSize(dev_id) -> 4;
regSize(eui) -> 8;
regSize(panadr) -> 4;
regSize(sys_cfg) -> 4;
regSize(sys_time) -> 5;
regSize(tx_fctrl) -> 5;
regSize(tx_buffer) -> 1024;
regSize(dx_time) -> 5;
regSize(rx_fwto) -> 2; % user manual gives 2 bytes and bits 16-31 are reserved
regSize(sys_ctrl) -> 4;
regSize(sys_mask) -> 4;
regSize(sys_status) -> 5;
regSize(rx_finfo) -> 4;
regSize(rx_buffer) -> 1024;
regSize(rx_fqual) -> 8;
regSize(rx_ttcki) -> 4;
regSize(rx_ttcko) -> 5;
```

```
regSize(rx_time) -> 14;
regSize(tx_time) -> 10;
regSize(tx_antd) -> 2;
regSize(sys_state) -> 4;
regSize(ack_resp_t) -> 4;
regSize(rx_sniff) -> 4;
regSize(tx_power) -> 4;
regSize(chan_ctrl) -> 4;
regSize(usr_sfd) -> 41;
regSize(agc_ctrl) -> 33;
regSize(ext_sync) -> 12;
regSize(acc_mem) -> 4064;
regSize(gpio_ctrl) -> 44;
regSize(drx_conf) -> 44; % user manual gives 44 bytes but sum of register length
    gives 45 bytes
regSize(rf_conf) -> 58; % user manual gives 58 but sum of all its register gives
    53 => Placeholder for the remaining 8 bytes
regSize(tx_cal) -> 13; % user manual gives 52 bytes but sum of all sub regs gives
    13 bytes
regSize(fs_ctrl) -> 21;
regSize(aon) -> 12;
regSize(otp_if) -> 19; % user manual gives 18 bytes in regs table but sum of all
    sub regs is 19 bytes
regSize(lde_ctrl) -> undefined; % No size ?
regSize(lde_if) -> undefined; % No size ?
regSize(dig_diag) -> 38; % user manual gives 41 bytes but sum of all sub regs
    gives 38 bytes
regSize(pmsc) -> 44. % user manual gives 48 bytes but sum of all sub regs gives 41
     bytes

%% Gives the size in bytes
subRegSize(agc_ctrl1) -> 2;
subRegSize(agc_tune1) -> 2;
subRegSize(agc_tune2) -> 4;
subRegSize(agc_tune3) -> 2;
subRegSize(agc_stat1) -> 3;
subRegSize(ec_ctrl) -> 4;
subRegSize(gpio_mode) -> 4;
subRegSize(gpio_dir) -> 4;
subRegSize(gpio_dout) -> 4;
subRegSize(gpio_irqe) -> 4;
subRegSize(gpio_isen) -> 4;
subRegSize(gpio_imode) -> 4;
subRegSize(gpio_ibes) -> 4;
subRegSize(gpio_iclr) -> 4;
subRegSize(gpio_idbe) -> 4;
subRegSize(gpio_raw) -> 4;
subRegSize(drx_tune0b) -> 2;
subRegSize(drx_tune1a) -> 2;
subRegSize(drx_tune1b) -> 2;
subRegSize(drx_tune2) -> 4;
subRegSize(drx_sfdtoc) -> 2;
subRegSize(drx_pretoc) -> 2;
subRegSize(drx_tune4h) -> 2;
subRegSize(rf_conf) -> 4;
subRegSize(rf_rxctrlh) -> 1;
subRegSize(rf_txctrl) -> 4; % ! table in user manual gives 3 but details gives 4
subRegSize(ldotune) -> 5;
subRegSize(tc_sarc) -> 2;
subRegSize(tc_pg_ctrl) -> 1;
subRegSize(tc_pgdelay) -> 1;
subRegSize(tc_pgtest) -> 1;
```

```
subRegSize(fs_pllcfg) -> 4;
subRegSize(fs_plltune) -> 1;
subRegSize(fs_xtalt) -> 1;
subRegSize(aon_wcfg) -> 2;
subRegSize(aon_ctrl) -> 1;
subRegSize(aon_rdat) -> 1;
subRegSize(aon_addr) -> 1;
subRegSize(aon_cfg0) -> 4;
subRegSize(aon_cfg1) -> 2;
subRegSize(otp_wdat) -> 4;
subRegSize(otp_addr) -> 2;
subRegSize(otp_ctrl) -> 2;
subRegSize(otp_stat) -> 2;
subRegSize(otp_rdat) -> 4;
subRegSize(otp_srdat) -> 4;
subRegSize(otp_sf) -> 1;
subRegSize(lde_thresh) -> 2;
subRegSize(lde_cfg1) -> 1;
subRegSize(lde_ppindx) -> 2;
subRegSize(lde_ppampl) -> 2;
subRegSize(lde_rxantd) -> 2;
subRegSize(lde_cfg2) -> 2;
subRegSize(lde_repc) -> 2;
subRegSize(evc_ctrl) -> 4;
subRegSize(diag_tmc) -> 2;
subRegSize(pmsc_ctrl0) -> 4;
subRegSize(pmsc_ctrl1) -> 4;
subRegSize(pmsc_snozt) -> 1;
subRegSize(pmsc_txfseq) -> 2;
subRegSize(pmsc_ledc) -> 4;
subRegSize(_) -> error({error}).

%--- Debug ----------------------------------------------------------------------

debug_read(Reg, Value) ->
    io:format("[PmodUWB] read [16#~2.16.0B - ~w] --> ~s -> ~s~n",
        [regFile(Reg), Reg, debug_bitstring(Value), debug_bitstring_hex(Value)]
    ).

debug_write(Reg, Value) ->
    io:format("[PmodUWB] write [16#~2.16.0B - ~w] --> ~s -> ~s~n",
        [regFile(Reg), Reg, debug_bitstring(Value), debug_bitstring_hex(Value)]
    ).
debug_write(Reg, SubReg, Value) ->
    io:format("[PmodUWB] write [16#~2.16.0B - ~w - 16#~2.16.0B - ~w] --> ~s -> ~s~
        n",
        [regFile(Reg), Reg, subReg(SubReg), SubReg, debug_bitstring(Value),
            debug_bitstring_hex(Value)]
    ).

debug_bitstring(Bitstring) ->
    lists:flatten([io_lib:format("2#~8.2.0B ", [X]) || <<X>> <= Bitstring]).

debug_bitstring_hex(Bitstring) ->
    lists:flatten([io_lib:format("16#~2.16.0B ", [X]) || <<X>> <= Bitstring]).
```

```
-module(pmod_uwb).
-behaviour(gen_server).

% API
-export([start_link/2]).
```

```erlang
-export([read/1, write/2, write_tx_data/1, get_received_data/0, transmit/1,
    transmit/2, wait_for_transmission/0, reception/0, reception/1]).
-export([reception_async/0]).
-export([set_frame_timeout/1]).
-export([set_preamble_timeout/1, disable_preamble_timeout/0]).
-export([softreset/0, clear_rx_flags/0]).
-export([disable_rx/0]).
-export([suspend_frame_filtering/0, resume_frame_filtering/0]).
-export([signal_power/0]).
-export([prf_value/0]).
-export([rx_preamble_repetition/0]).
-export([rx_data_rate/0]).
-export([rx_ranging_info/0]).
-export([std_noise/0]).
-export([first_path_power_level/0]).
-export([get_conf/0]).
-export([get_rx_metadata/0]).

% gen_server callback
-export([init/1, handle_call/3, handle_cast/2]).

-compile({nowarn_unused_function, [debug_read/2, debug_write/2, debug_write/3,
    debug_bitstring/1, debug_bitstring_hex/1]}).

% Includes
-include("grisp.hrl").

-include("pmod_uwb.hrl").

%--- Macros -------------------------------------------------------------------

% Define the polarity and the phase of the clock
-define(SPI_MODE, #{clock => {low, leading}}).

-define(WRITE_ONLY_REG_FILE(RegFileID), RegFileID == tx_buffer).

-define(READ_ONLY_REG_FILE(RegFileID), RegFileID==dev_id;
                                       RegFileID==sys_time;
                                       RegFileID==rx_finfo;
                                       RegFileID==rx_buffer;
                                       RegFileID==rx_fqual;
                                       RegFileID==rx_ttcko;
                                       RegFileID==rx_time;
                                       RegFileID==tx_time;
                                       RegFileID==sys_state;
                                       RegFileID==acc_mem).

%% The congifurations of the subregisters of these register files are different
%% (some sub-registers are RO, some are RW and some have reserved bytes
%% that can't be written)
%% Thus, some registers files require to write their sub-register independently
%% => Write the sub-registers one by one instead of writting
%%    the whole register file directly
-define(IS_SRW(RegFileID), RegFileID==agc_ctrl;
                           RegFileID==ext_sync;
                           RegFileID==ec_ctrl;
                           RegFileID==gpio_ctrl;
                           RegFileID==drx_conf;
                           RegFileID==rf_conf;
                           RegFileID==tx_cal;
                           RegFileID==fs_ctrl;
                           RegFileID==aon;
```

```
                             RegFileID==otp_if;
                             RegFileID==lde_if;
                             RegFileID==dig_diag;
                             RegFileID==pmsc).

-define(READ_ONLY_SUB_REG(SubRegister), SubRegister==irqs;
                                           SubRegister==agc_stat1;
                                           SubRegister==ec_rxtc;
                                           SubRegister==ec_glop;
                                           SubRegister==drx_car_int;
                                           SubRegister==rf_status;
                                           SubRegister==tc_sarl;
                                           SubRegister==sarw;
                                           SubRegister==tc_pg_status;
                                           SubRegister==lde_thresh;
                                           SubRegister==lde_ppindx;
                                           SubRegister==lde_ppampl;
                                           SubRegister==evc_phe;
                                           SubRegister==evc_rse;
                                           SubRegister==evc_fcg;
                                           SubRegister==evc_fce;
                                           SubRegister==evc_ffr;
                                           SubRegister==evc_ovr;
                                           SubRegister==evc_sto;
                                           SubRegister==evc_pto;
                                           SubRegister==evc_fwto;
                                           SubRegister==evc_txfs;
                                           SubRegister==evc_hpw;
                                           SubRegister==evc_tpw).


%--- Types ---------------------------------------------------------------------
-export_type([register_values/0]).

-type regFileID() :: atom().
-opaque register_values() :: map().

%--- API -----------------------------------------------------------------------

start_link(Connector, _Opts) ->
    gen_server:start_link({local, ?MODULE}, ?MODULE, Connector, []).


%% @doc read a register file
%%
%% === Example ===
%% To read the register file DEV_ID
%% ```
%% 1> pmod_uwb:read(dev_id).
%% #{model => 1,rev => 0,ridtag => "DECA",ver => 3}
%% '''
-spec read(RegFileID) -> Result when
    RegFileID :: regFileID(),
    Result    :: map() | {error, any()}.
read(RegFileID) when ?WRITE_ONLY_REG_FILE(RegFileID) ->
    error({read_on_write_only_register, RegFileID});
read(RegFileID) -> call({read, RegFileID}).

%% @doc Write values in a register
%%
%% === Examples ===
%% To write in a simple register file (i.e. a register without any sub-register)
```

```
%% ```
%% 1> pmod_uwb:write(eui, #{eui => <<16#AAAAAABBBBBBBBBB>>}).
%% ok
%% '''
%% To write in one sub-register of a register file:
%% ```
%% 2> pmod_uwb:write(panadr, #{pan_id => <<16#AAAA>>}).
%% ok
%% '''
%% The previous code will only change the values inside the sub-register PAN_ID
%%
%% To write in multiple sub-register of a register file in the same burst:
%% ```
%% 3> pmod_uwb:write(panadr, #{pan_id => <<16#AAAA>>,
%%                             short_addr => <<16#BBBB>>}).
%% ok
%% '''
%% Some sub-registers have their own fields. For example to set the value of
%% the DIS_AM field in the sub-register AGC_CTRL1 of the register file AGC_CTRL:
%% ```
%% 4> pmod_uwb:write(agc_ctrl, #{agc_ctrl1 => #{dis_am => 2#0}}).
%% '''
-spec write(RegFileID, Value) -> Result when
    RegFileID :: regFileID(),
    Value     :: map(),
    Result    :: ok | {error, any()}.
write(RegFileID, Value) when ?READ_ONLY_REG_FILE(RegFileID) ->
    error({write_on_read_only_register, RegFileID, Value});
write(RegFileID, Value) when is_map(Value) ->
    call({write, RegFileID, Value}).

%% @doc Writes the data in the TX_BUFFER register
%%
%% Value is expected to be a <b>Binary</b>
%% That choice was made to make the transmission of frames easier later on
%%
%% === Examples ===
%% Send "Hello" in the buffer
%% ```
%% 1> pmod_uwb:write_tx_data(<<"Hello">>).
%% '''
-spec write_tx_data(Value) -> Result when
    Value  :: binary(),
    Result :: ok | {error, any()}.
write_tx_data(Value) -> call({write_tx, Value}).

%% @doc Retrieves the data received on the UWB antenna
%% @returns {DataLength, Data}
-spec get_received_data() -> Result when
    Result :: {integer(), bitstring()} | {error, any()}.
get_received_data() -> call({get_rx_data}).

get_rx_metadata() ->
    #{rng := Rng} = read(rx_finfo),
    #{rx_stamp := RxStamp} = read(rx_time),
    #{tx_stamp := TxStamp} = read(tx_time),
    #{rxtofs := Rxtofs} = read(rx_ttcko),
    #{rxttcki := Rxttcki} = read(rx_ttcki),
    #{snr => snr(),
      prf => prf_value(),
      pre => rx_preamble_repetition(),
      data_rate => rx_data_rate(),
```

```
      rng => Rng,
      rx_stamp => RxStamp,
      tx_stamp => TxStamp,
      rxtofs => Rxtofs,
      rxttcki => Rxttcki}.

% Source: https://forum.qorvo.com/t/how-to-calculate-the-signal-to-noise-ratio-snr
    -of-dw1000/5585/3
snr() ->
    Delta = 87-7.5,
    RSL = pmod_uwb:signal_power(),
    RSL + Delta.

%% @doc Transmit data with the default options (i.e. don’t wait for resp, ...)
%%
%% === Examples ===
%% To transmit a frame:
%% ‘‘‘
%% 1> pmod_uwb:transmit(<Version:4, NextHop:8>>).
%% ok.
%% ’’’
-spec transmit(Data) -> Result when
    Data   :: bitstring(),
    Result :: ok.
transmit(Data) when is_bitstring(Data) ->
    call({transmit, Data, #tx_opts{}}),
    wait_for_transmission().

%% @doc Performs a transmission with the specified options
%%
%% === Options ===
%% * wait4resp: It specifies that the reception must be enabled after
%%              the transmission in the expectation of a response
%% * w4r-tim: Specifies the turn around time in microseconds. That is the time
%%            the pmod will wait before enabling rx after a tx.
%%            Note that it won’t be set if wit4resp is disabled
%% * txdlys: Specifies if the transmitter delayed sending should be set
%% * tx_delay: Specifies the delay of the transmission (see register DX_TIME)
%%
%% === Examples ===
%% To transmit a frame with default options:
%% ‘‘‘
%% 1> pmod_uwb:transmit(<Version:4, NextHop:8>>, #tx_opts{}).
%% ok.
%% ’’’
-spec transmit(Data, Options) -> Result when
    Data :: bitstring(),
    Options :: tx_opts(),
    Result :: ok.
transmit(Data, Options) ->
    case Options#tx_opts.wait4resp of
        ?ENABLED -> clear_rx_flags();
        _ -> ok
    end,
    call({transmit, Data, Options}),
    case read(sys_status) of
        #{hdpwarn := 2#1} -> error({hdpwarn});
        _ -> ok
    end,
    wait_for_transmission().

%% Wait for the transmission to be performed
```

```
%% usefull in the case of a delayed transmission
wait_for_transmission() ->
    case read(sys_status) of
        #{txfrs := 1} -> ok;
        _ -> wait_for_transmission()
    end.

%% @doc Receive data using the pmod
%% @equiv reception(false)
-spec reception() -> Result when
    Result :: {integer(), bitstring()} | {error, any()}.
reception() ->
    reception(false).

%% @doc Receive data using the pmod
%%
%% The function will hang until a frame is received on the board
%%
%% The CRC of the received frame <b>isn't</b> included in the returned value
%%
%% @param RXEnabled: specifies if the reception is already enabled on the board
%%                   (or set with delay)
%%
%% === Example ===
%% ```
%% 1> pmod_uwb:reception().
%% % Some frame is transmitted
%% {11, <<"Hello world">>}.
%% '''
-spec reception(RXEnabled) -> Result when
      RXEnabled :: boolean(),
      Result    :: {integer(), bitstring()} | {error, any()}.
reception(RXEnabled) ->
    if not RXEnabled -> enable_rx();
       true -> ok
    end,
    case wait_for_reception() of
        ok ->
            get_received_data();
        Err ->
            {error, Err}
    end.

-spec reception_async() -> Result when
      Result    :: ok | {error, any()}.
reception_async() ->
    case reception() of
        {error, _} = Err -> Err;
        Frame ->
            Metadata = get_rx_metadata(),
            ieee802154_events:rx_event(Frame, Metadata)
    end.

%% @private
enable_rx() ->
    % io:format("Enabling reception~n"),
    clear_rx_flags(),
    call({write, sys_ctrl, #{rxenab => 2#1}}).

%% @doc Disables the reception on the pmod
disable_rx() ->
    call({write, sys_ctrl, #{trxoff => 2#1}}).
```

```erlang

wait_for_reception() ->
    % io:format("Wait for resp~n"),
    case read(sys_status) of
        #{rxrfto := 1} -> rxrfto;
        #{rxphe := 1} -> rxphe;
        #{rxfce := 1} -> rxfce;
        #{rxrfsl := 1} -> rxrfsl;
        #{rxpto := 1} -> rxpto;
        #{rxsfdto := 1} -> rxsfdto;
        #{ldeerr := 1} -> ldeerr;
        #{affrej := 1} -> affrej;
        #{rxdfr := 0} -> wait_for_reception();
        #{rxfce := 1} -> rxfce;
        #{rxfcg := 1} -> ok;
        #{rxfcg := 0} -> wait_for_reception();
        % #{rxdfr := 1, rxfcg := 1} -> ok; % The example driver doesn't do that but the user manual says that how you should check the reception of a frame
        _ -> error({error_wait_for_reception})
    end.

%% @doc Set the frame wait timeout and enables it
%% The unit is roughly 1us (cf. user manual)
%% If a float is given, it's decimal part is removed using trunc/1
%% @end
-spec set_frame_timeout(Timeout) -> Result when
      Timeout :: microseconds(),
      Result  :: ok.
set_frame_timeout(Timeout) when is_float(Timeout) ->
    set_frame_timeout(trunc(Timeout));
set_frame_timeout(Timeout) when is_integer(Timeout) ->
    write(rx_fwto, #{rxfwto => Timeout}),
    write(sys_cfg, #{rxwtoe => 2#1}). % enable receive wait timeout

%% @doc Sets the preamble timeout. (PRETOC register of the DW1000)
%% The unit of 'Timeout' is in units usec
%% If the value is a float, trunc is called to remove the decimal part
%% Internally, it's converted in untis of PAC size
-spec set_preamble_timeout(Timeout) -> ok when
      Timeout :: non_neg_integer().
set_preamble_timeout(TO) when is_float(TO) ->
    set_preamble_timeout(trunc(TO));
set_preamble_timeout(TO) when is_integer(TO) ->
    call({preamble_timeout, TO}),
    write(drx_conf, #{drx_pretoc => 0}).

disable_preamble_timeout() ->
    write(drx_conf, #{drx_pretoc => 0}).

%% @doc Performs a reset of the IC following the procedure (cf. sec. 7.2.50.1)
softreset() ->
    write(pmsc, #{pmsc_ctrl0 => #{sysclks => 2#01}}),
    write(pmsc, #{pmsc_ctrl0 => #{softrest => 16#0}}),
    write(pmsc, #{pmsc_ctrl0 => #{softreset => 16#FFFF}}).


clear_rx_flags() ->
    write(sys_status, #{rxsfdto => 2#1,
                        rxpto => 2#1,
                        rxrfto => 2#1,
                        rxrfsl => 2#1,
```

```
                                rxfce => 2#1,
                                rxphe => 2#1,
                                rxprd => 2#1,
                                rxdsfdd => 2#1,
                                rxphd => 2#1,
                                rxdfr => 2#1,
                                rxfcg => 2#1}).

suspend_frame_filtering() ->
    write(sys_cfg, #{ffen => 2#0}).

resume_frame_filtering() ->
    write(sys_cfg, #{ffen => 2#1}).

%% @doc Returns the estimated value of the signal power in dBm
%% cf. user manual section 4.7.2
signal_power() ->
    C = channel_impulse_resp_pow() , % Channel impulse resonse power value (
        CIR_PWR)
    A = case prf_value() of
            16 -> 113.77;
            64 -> 121.74
        end, % Constant. For PRF of 16 MHz = 113.77, for PRF of 64MHz = 121.74
    N = preamble_acc(), % Preamble accumulation count value (RXPACC but might be
        ajusted)
    % io:format("C: ~w~n A:~w~n N:~w~n", [C, A, N]),
    Res = 10 * math:log10((C* math:pow(2, 17))/math:pow(N, 2)) - A,
    % io:format("Estimated signal power: ~p dBm~n", [Res]),
    % io:format("Std noise: ~w~n", [pmod_uwb:read(rx_fqual)]),
    Res.

preamble_acc() ->
    #{rxpacc := RXPACC} = read(rx_finfo),
    #{rxpacc_nosat := RXPACC_NOSAT} = read(drx_conf),
    if
        RXPACC == RXPACC_NOSAT -> RXPACC - 5;
        true -> RXPACC
    end.

channel_impulse_resp_pow() ->
    #{cir_pwr := CIR_PWR} = read(rx_fqual),
    CIR_PWR.

%% @doc Gives the value of the PRF in MHz
-spec prf_value() -> 16 | 64.
prf_value() ->
    #{agc_tune1 := AGC_TUNE1} = read(agc_ctrl),
    case AGC_TUNE1 of
        16#8870 -> 16;
        16#889B -> 64
    end.

%% @doc returns the preamble symbols repetition
rx_preamble_repetition() ->
    #{rxpsr := RXPSR} = read(rx_finfo),
    case RXPSR of
        0 -> 16;
        1 -> 64;
        2 -> 1024;
        3 -> 4096
    end.
```

```
%% @doc returns the data rate of the received frame in kbps
rx_data_rate() ->
    #{rxbr := RXBR} = read(rx_finfo),
    case RXBR of
        0 -> 110;
        1 -> 850;
        2 -> 6800
    end.

% @doc returns the value of the 'Ranging' bit of the received frame
rx_ranging_info() ->
    #{rng := RNG} = read(rx_finfo),
    RNG.

std_noise() ->
    #{std_noise := STD_NOISE} = read(rx_fqual),
    STD_NOISE.

first_path_power_level() ->
    #{fp_ampl1 := F1} = read(rx_time),
    #{fp_ampl2 := F2, pp_ampl3 := F3} = read(rx_fqual),
    A = 113.77,
    N = preamble_acc(),
    10 * math:log10((math:pow(F1,2) + math:pow(F2, 2) + math:pow(F3, 2))/math:pow(
        N, 2)) - A.

get_conf() ->
    call({get_conf}).

%--- gen_server Callbacks -----------------------------------------------------

%% @private
init(Slot) ->
    % Verify the slot used
    case {grisp_hw:platform(), Slot} of
        {grisp2, spi2} -> ok;
        {P, S} -> error({incompatible_slot, P, S})
    end,
    grisp_devices:register(Slot, ?MODULE),
    Bus = grisp_spi:open(Slot),
    case verify_id(Bus) of
        ok -> softreset(Bus);
        Val -> error({dev_id_no_match, Val})
    end,
    ldeload(Bus),
    % TODO Merge the next 4 cfg commands into one
    write_default_values(Bus),
    config(Bus),
    setup_sfd(Bus),
    Conf =  #phy_cfg{},
    {ok, #{bus => Bus, conf => Conf}}.

%% @private
handle_call({read, RegFileID}, _From, #{bus := Bus} = State)              ->
    {reply, read_reg(Bus, RegFileID), State};
handle_call({write, RegFileID, Value}, _From, #{bus := Bus} = State)       ->
    {reply, write_reg(Bus, RegFileID, Value), State};
handle_call({write_tx, Value}, _From, #{bus := Bus} = State)              ->
    {reply, write_tx_data(Bus, Value), State};
handle_call({transmit, Data, Options}, _From, #{bus := Bus} = State)       ->
    {reply, tx(Bus, Data, Options), State};
handle_call({delayed_transmit, Data, Delay}, _From, #{bus := Bus} = State) ->
```

```erlang
    {reply, delayed_tx(Bus, Data, Delay), State};
handle_call({get_rx_data}, _From, #{bus := Bus} = State)                     ->
    {reply, get_rx_data(Bus), State};
handle_call({get_conf}, _From, #{conf := Conf} = State)                      ->
    {reply, Conf, State};
handle_call({preamble_timeout, TOus}, _From, State)                          ->
    #{bus := Bus, conf := Conf} = State,
    PACSize = Conf#phy_cfg.pac_size,
    case TOus of
        0 ->
            write_reg(Bus, drx_conf, #{drx_pretoc => 0});
        _ ->
            % Remove 1 because DW1000 counter auto. adds 1 (cf. 7.2.40.9 user manual)
            To = math:ceil(TOus / PACSize)-1,
            write_reg(Bus, drx_conf, #{drx_pretoc => round(To)})
    end,
    {reply, ok, State};
handle_call(Request, _From, _State)                                          ->
    error({unknown_call, Request}).

%% @private
handle_cast(Request, _State) -> error({unknown_cast, Request}).

%--- Internal -----------------------------------------------------------------

call(Call) ->
    Dev = grisp_devices:default(?MODULE),
    gen_server:call(Dev#device.pid, Call).


%% @doc Varify the dev_id register of the pmod
%% @returns ok if the value is correct, otherwise the value read
verify_id(Bus) ->
    #{ridtag := RIDTAG, model := MODEL} = read_reg(Bus, dev_id),
    case {RIDTAG, MODEL} of
        {"DECA", 1} -> ok;
        _ -> {RIDTAG, MODEL}
    end.

%% @private
%% Performs a softreset on the pmod
-spec softreset(Bus::grisp_spi:ref()) -> ok.
softreset(Bus) ->
    write_reg(Bus, pmsc, #{pmsc_ctrl0 => #{sysclks => 2#01}}),
    write_reg(Bus, pmsc, #{pmsc_ctrl0 => #{softrest => 16#0}}),
    write_reg(Bus, pmsc, #{pmsc_ctrl0 => #{softreset => 16#FFFF}}).

%% @private
%% Writes the default values described in section 2.5.5 of the user manual
-spec write_default_values(Bus::grisp_spi:ref()) -> ok.
write_default_values(Bus) ->
    write_reg(Bus, lde_if, #{lde_cfg1 => #{ntm => 16#D}, lde_cfg2 => 16#1607}),
    write_reg(Bus, agc_ctrl, #{agc_tune1 => 16#8870, agc_tune2 => 16#2502A907}),
    write_reg(Bus, drx_conf, #{drx_tune2 => 16#311A002D}),
    write_reg(Bus, tx_power, #{tx_power => 16#0E082848}),
    write_reg(Bus, rf_conf, #{rf_txctrl => 16#001E3FE3}),
    write_reg(Bus, tx_cal, #{tc_pgdelay => 16#B5}),
    write_reg(Bus, fs_ctrl, #{fs_plltune => 16#BE}).

%% @private
config(Bus) ->
```

```
    write_reg(Bus, ext_sync, #{ec_ctrl => #{pllldt => 2#1}}),
    %write_reg(Bus, pmsc, #{pmsc_ctrl1 => #{lderune => 2#0}}),
    % Now enable RX and TX leds
    write_reg(Bus, gpio_ctrl, #{gpio_mode => #{msgp2 => 2#01, msgp3 => 2#01}}),
    % Enable RXOK and SFD leds
    write_reg(Bus, gpio_ctrl, #{gpio_mode => #{msgp0 => 2#01, msgp1 => 2#01}}),
    write_reg(Bus, pmsc, #{pmsc_ctrl0 => #{gpdce => 2#1, khzclken => 2#1}}),
    write_reg(Bus, pmsc, #{pmsc_ledc => #{blnken => 2#1}}),
    write_reg(Bus, dig_diag, #{evc_ctrl => #{evc_en => 2#1}}), % enable counting event for debug purposes
    % write_reg(Bus, sys_cfg, #{rxwtoe => 2#1}),
    write_reg(Bus, tx_fctrl, #{txpsr => 2#10}). % Setting preamble symbols to 1024

%% @private
%% Load the microcode from ROM to RAM
%% It follows the steps described in section 2.5.5.10 of the DW1000 user manual
ldeload(Bus) ->
    write_reg(Bus, pmsc, #{pmsc_ctrl0 => #{sysclks => 2#01}}),
    write_reg(Bus, pmsc, #{pmsc_ctrl0 => #{otp => 2#1, res8 => 2#1}}), % Writes 0x0301 in pmsc_ctrl0
    write_reg(Bus, otp_if, #{otp_ctrl => #{ldeload => 2#1}}), % Writes 0x8000 in OTP_CTRL
    timer:sleep(150), % User manual requires a wait of 150 s
    write_reg(Bus, pmsc, #{pmsc_ctrl0 => #{sysclks => 2#0}}), % Writes 0x0200 in pmsc_ctrl0
    write_reg(Bus, pmsc, #{pmsc_ctrl0 => #{res8 => 2#0}}).

%% @private
%% If no frame is transmitted before AUTOACK, then the SFD isn't properly set
%% (cf. section 5.3.1.2 SFD initialisation)
setup_sfd(Bus) ->
    write_reg(Bus, sys_ctrl, #{txstrt => 2#1, trxoff => 2#1}).

%% @private
%% Transmit the data using UWB
%% @param Options is used to set options about the transmission like a transmission delay, etc.
-spec tx(grisp_spi:ref(), Data :: binary(), Options :: #tx_opts{}) -> ok.
tx(Bus, Data, #tx_opts{wait4resp = Wait4resp, w4r_tim = W4rTim, txdlys = TxDlys, tx_delay = TxDelay, ranging = Ranging}) ->
    % Writing the data that will be sent (w/o CRC)
    DataLength = byte_size(Data) + 2, % DW1000 automatically adds the 2 bytes CRC
    write_tx_data(Bus, Data),
    % Setting the options of the transmission
    case Wait4resp of
        ?ENABLED -> write_reg(Bus, ack_resp_t, #{w4r_tim => W4rTim});
        _ -> ok
    end,
    case TxDlys of
        ?ENABLED -> write_reg(Bus, dx_time, #{dx_time => TxDelay});
        _ -> ok
    end,
    write_reg(Bus, tx_fctrl, #{txboffs => 2#0, tr => Ranging, tflen => DataLength}),
    write_reg(Bus, sys_ctrl, #{txstrt => 2#1, wait4resp => Wait4resp, txdlys => TxDlys}). % start transmission and some options

%% @private
%% Transmit the data with a specified delay using UWB
delayed_tx(Bus, Data, Delay) ->
    write_reg(Bus, dx_time, #{dx_time => Delay}),
    DataLength = byte_size(Data) + 2, % DW1000 automatically adds the 2 bytes CRC
```

```
    write_tx_data(Bus, Data),
    write_reg(Bus, tx_fctrl, #{txboffs => 2#0, tflen => DataLength}),
    write_reg(Bus, sys_ctrl, #{txstrt => 2#1, txdlys => 2#1}). % start transmission

%% @private
%% Get the received data (without the CRC bytes) stored in the rx_buffer
get_rx_data(Bus) ->
    #{rxflen := FrameLength} = read_reg(Bus, rx_finfo),
    Frame = read_rx_data(Bus, FrameLength-2), % Remove the CRC bytes
    {FrameLength, Frame}.

%% @private
%% @doc Reverse the byte order of the bitstring given in the argument
%% @param Bin a bitstring
reverse(Bin) -> reverse(Bin, <<>>).
reverse(<<Bin:8>>, Acc) ->
    <<Bin, Acc/binary>>;
reverse(<<Bin:8, Rest/bitstring>>, Acc) ->
    reverse(Rest, <<Bin, Acc/binary>>).

% Source: https://stackoverflow.com/a/43310493
% reverse(Binary) ->
%     Size = bit_size(Binary),
%     <<X:Size/integer-little>> = Binary,
%     <<X:Size/integer-big>>.

%% @private
%% @doc Creates the header of the SPI transaction between the GRiSP and the pmod
%%
%%  It creates a header of 1 bytes. The header is used in a transaction that will affect
%%  the whole register file (read/write)
%%
%% @param Op an atom (either <i>read</i> or <i>write</i>
%% @param RegFileID an atom representing the register file
%% @returns a formated header of <b>1 byte</b> long as described in the user manual
header(Op, RegFileID) ->
    <<(rw(Op)):1, 2#0:1, (regFile(RegFileID)):6>>.

%% @private
%% @doc Creates the header of the SPI transaction between the GRiSP and the pmod
%%
%%  It creates a header of 2 bytes. The header is used in a transaction that will affect
%%  the whole sub-register (read/write)
%%  Careful: The sub-register needs to be mapped in the hrl file
%%
%% @param Op an atom (either <i>read</i> or <i>write</i>
%% @param RegFileID an atom representing the register file
%% @param SubRegister an atom representing the sub-register
%% @returns a formated header of <b>2 byte</b> long as described in the user manual
header(Op, RegFileID, SubRegister) ->
    case subReg(SubRegister) < 127 of
        true -> header(Op, RegFileID, SubRegister, 2);
        _ -> header(Op, RegFileID, SubRegister, 3)
    end.

header(Op, RegFileID, SubRegister, 2) ->
    << (rw(Op)):1, 2#1:1, (regFile(RegFileID)):6,
```

```
        2#0:1, (subReg(SubRegister)):7 >>;
header(Op, RegFileID, SubRegister, 3) ->
    <<_:1, HighOrder:8, LowOrder:7>> = <<(subReg(SubRegister)):16>>,
    << (rw(Op)):1, 2#1:1, (regFile(RegFileID)):6,
       2#1:1, LowOrder:7,
       HighOrder:8>>.

%% @private
%% @doc Read the values stored in a register file
read_reg(Bus, lde_ctrl) -> read_reg(Bus, lde_if);
read_reg(Bus, lde_if) ->
    lists:foldl(fun(Elem, Acc) ->
                    Res = read_sub_reg(Bus, lde_if, Elem),
                    maps:merge(Acc, Res)
                end,
                #{},
                [lde_thresh, lde_cfg1, lde_ppindx, lde_ppampl, lde_rxantd,
                    lde_cfg2, lde_repc]);
read_reg(Bus, RegFileID) ->
    Header = header(read, RegFileID),
    [Resp] = grisp_spi:transfer(Bus, [{?SPI_MODE, Header, 1, regSize(RegFileID)}])
        ,
    % debug_read(RegFileID, Resp),
    reg(decode, RegFileID, Resp).


read_sub_reg(Bus, RegFileID, SubRegister) ->
    Header = header(read, RegFileID, SubRegister),
    HeaderSize = byte_size(Header),
    % io:format("[HEADER] type ~w - ~w - ~w~n", [HeaderSize, Header, subRegSize(
        SubRegister)]),
    [Resp] = grisp_spi:transfer(Bus, [{?SPI_MODE, Header, HeaderSize, subRegSize(
        SubRegister)}]),
    reg(decode, SubRegister, Resp).


%% @doc get the received data
%% @param Length is the total length of the data we are trying to read
read_rx_data(Bus, Length) ->
    Header = header(read, rx_buffer),
    [Resp] = grisp_spi:transfer(Bus, [{?SPI_MODE, Header, 1, Length}]),
    Resp.

% TODO: check that user isn't trying to write reserved bits by passing res, res1,
    ... in the map fields
%% @doc used to write the values in the map given in the Value argument
-spec write_reg(Bus::grisp_spi:ref(), RegFileID::regFileID(), Value::map()) -> ok.
% Write each sub-register one by one.
% If the user tries to write in a read-only sub-register, an error is thrown
write_reg(Bus, RegFileID, Value) when ?IS_SRW(RegFileID) ->
    maps:map(
        fun(SubRegister, Val) ->
            CurrVal = maps:get(SubRegister, read_reg(Bus, RegFileID)), % ? can the
                 read be done before ? Maybe but not assured that no values
                changes after a write in the register
            Body = case CurrVal of
                        V when is_map(V) -> reg(encode, SubRegister, maps:
                            merge_with(fun(_Key, _Old, New) -> New end, CurrVal,
                            Val));
                        _ -> reg(encode, SubRegister, #{SubRegister => Val})
                   end,
            Header = header(write, RegFileID, SubRegister),
```

```
            % debug_write(RegFileID, SubRegister, Body),
            _ = grisp_spi:transfer(Bus, [{?SPI_MODE, <<Header/binary, Body/binary
                >>, 2+subRegSize(SubRegister), 0}])
        end,
        Value),
    ok;
write_reg(Bus, RegFileID, Value) ->
    Header = header(write, RegFileID),
    CurrVal = read_reg(Bus, RegFileID),
    ValuesToWrite = maps:merge_with(fun(_Key, _Value1, Value2) -> Value2 end,
        CurrVal, Value),
    Body = reg(encode, RegFileID, ValuesToWrite),
    % debug_write(RegFileID, Body),
    _ = grisp_spi:transfer(Bus, [{?SPI_MODE, <<Header/binary, Body/binary>>, 1+
        regSize(RegFileID), 0}]),
    ok.

%% @doc write_tx_data/2 sends data (Value) in the register tx_buffer
%% @param Value is the data to be written. It must be a binary and have a size of
    maximum 1024 bits
write_tx_data(Bus, Value) when is_binary(Value), (bit_size(Value) < 1025) ->
    Header = header(write, tx_buffer),
    Length = byte_size(Value),
    % debug_write(tx_buffer, Body),
    _ = grisp_spi:transfer(Bus, [{?SPI_MODE, <<Header/binary, Value/binary>>, 1+
        Length, 0}]),
    ok.

%---- Register mapping -------------------------------------------------------

%% @doc Used to either decode the data returned by the pmod or to encode to data
    that will be sent to the pmod
%%
%% The transmission on the MISO line is done byte by byte starting from the lowest
     rank byte to the highest rank
%% Example: dev_id value is 0xDECA0130 but 0x3001CADE is transmitted over the MISO
     line
-spec reg(Type, Register, Val) -> Ret when
      Type     :: encode | decode,
      Register :: regFileID(),
      Val      :: nonempty_binary() | register_values(),
      Ret      :: nonempty_binary() | register_values().
reg(encode, SubRegister, Value) when ?READ_ONLY_SUB_REG(SubRegister) -> error({
    writing_read_only_sub_register, SubRegister, Value});
reg(decode, dev_id, Resp) ->
    <<
      RIDTAG:16, Model:8, Ver:4, Rev:4
    >> = reverse(Resp),
    #{
        ridtag => integer_to_list(RIDTAG, 16), model => Model, ver => Ver, rev =>
            Rev
    };
reg(decode, eui, Resp) ->
    #{
        eui => reverse(Resp)
    };
reg(encode, eui, Val) ->
    #{
        eui := EUI
    } = Val,
    reverse(
        EUI
```

```
    );
reg(decode, panadr, Resp) ->
    <<
        PanId:16, ShortAddr:16
    >> = reverse(Resp),
    #{
        pan_id => <<PanId:16>>, short_addr => <<ShortAddr:16>>
    };
reg(encode, panadr, Val) ->
    #{
        pan_id := PanId, short_addr := ShortAddr
    } = Val,
    reverse(<<
        PanId:16/bitstring, ShortAddr:16/bitstring
    >>);
reg(decode, sys_cfg, Resp) ->
    <<
        FFA4:1, FFAR:1, FFAM:1, FFAA:1, FFAD:1, FFAB:1, FFBC:1, FFEN:1, % bits 7-0
        FCS_INIT2F:1, DIS_RSDE:1, DIS_PHE:1, DIS_DRXB:1, DIS_FCE:1, SPI_EDGE:1, HIRQ_POL:1, FFA5:1, % bits 15-8
        _:1, RXM110K:1, _:3, DIS_STXP:1, PHR_MODE:2, % bits 23-16
        AACKPEND:1, AUTOACK:1, RXAUTR:1, RXWTOE:1, _:4 % bits 31-24
    >> = Resp,
    #{
        aackpend => AACKPEND, autoack => AUTOACK, rxautr => RXAUTR, rxwtoe => RXWTOE,
        rxm110k => RXM110K, dis_stxp => DIS_STXP, phr_mode => PHR_MODE,
        fcs_init2F => FCS_INIT2F, dis_rsde => DIS_RSDE, dis_phe => DIS_PHE, dis_drxb => DIS_DRXB, dis_fce => DIS_FCE, spi_edge => SPI_EDGE, hirq_pol => HIRQ_POL, ffa5 => FFA5,
        ffa4 => FFA4, ffar => FFAR, ffam => FFAM, ffaa => FFAA, ffad => FFAD, ffab => FFAB, ffbc => FFBC, ffen => FFEN
    };
reg(encode, sys_cfg, Val) ->
    #{
        aackpend := AACKPEND, autoack := AUTOACK, rxautr := RXAUTR, rxwtoe := RXWTOE,
        rxm110k := RXM110K, dis_stxp := DIS_STXP, phr_mode := PHR_MODE,
        fcs_init2F := FCS_INIT2F, dis_rsde := DIS_RSDE, dis_phe := DIS_PHE, dis_drxb := DIS_DRXB, dis_fce := DIS_FCE, spi_edge := SPI_EDGE, hirq_pol := HIRQ_POL, ffa5 := FFA5,
        ffa4 := FFA4, ffar := FFAR, ffam := FFAM, ffaa := FFAA, ffad := FFAD, ffab := FFAB, ffbc := FFBC, ffen := FFEN
    } = Val,
    <<
        FFA4:1, FFAR:1, FFAM:1, FFAA:1, FFAD:1, FFAB:1, FFBC:1, FFEN:1, % bits 7-0
        FCS_INIT2F:1, DIS_RSDE:1, DIS_PHE:1, DIS_DRXB:1, DIS_FCE:1, SPI_EDGE:1, HIRQ_POL:1, FFA5:1, % bits 15-8
        2#0:1, RXM110K:1, 2#0:3, DIS_STXP:1, PHR_MODE:2, % bits 23-16
        AACKPEND:1, AUTOACK:1, RXAUTR:1, RXWTOE:1, 2#0:4 % bits 31-24
    >>;
reg(decode, sys_time, Resp) ->
    <<
        SysTime:40
    >> = reverse(Resp),
    #{
        sys_time => SysTime
    };
reg(decode, tx_fctrl, Resp) ->
    <<
        IFSDELAY:8, TXBOFFS:10, PE:2, TXPSR:2, TXPRF:2, TR:1, TXBR:2, R:3, TFLE:3, TFLEN:7
```

```
    >> = reverse(Resp),
    #{
        ifsdelay => IFSDELAY, txboffs => TXBOFFS, pe => PE, txpsr => TXPSR, txprf => TXPRF, tr => TR, txbr => TXBR, r => R, tfle => TFLE, tflen => TFLEN
    };
reg(encode, tx_fctrl, Val) ->
    #{
        ifsdelay := IFSDELAY, txboffs := TXBOFFS, pe := PE, txpsr := TXPSR, txprf := TXPRF, tr := TR, txbr := TXBR, r := R, tfle := TFLE, tflen := TFLEN
    } = Val,
    reverse(<<
        IFSDELAY:8, TXBOFFS:10, PE:2, TXPSR:2, TXPRF:2, TR:1, TXBR:2, R:3, TFLE:3, TFLEN:7
    >>);
% TX_BUFFER is write only => no decode
reg(decode, dx_time, Resp) ->
    #{
        dx_time => reverse(Resp)
    };
reg(encode, dx_time, Val) ->
    #{
        dx_time := DX_TIME
    } = Val,
    reverse(<<
        DX_TIME:40
    >>);
reg(decode, rx_fwto, Resp) ->
    <<
        RXFWTO:16
    >> = reverse(Resp),
    #{
        rxfwto => RXFWTO
    };
reg(encode, rx_fwto, Val) ->
    #{
        rxfwto := RXFWTO
    } = Val,
    reverse(<<
        RXFWTO:16
    >>);
reg(decode, sys_ctrl, Resp) ->
    <<
        WAIT4RESP:1, TRXOFF:1, _:2, CANSFCS:1, TXDLYS:1, TXSTRT:1, SFCST:1, % bits 7-0
        _:6, RXDLYE:1, RXENAB:1, % bits 15-8
        _:8, % bits 23-16
        _:7, HRBPT:1 % bits 31-24
    >> = Resp,
    #{
        sfcst => SFCST, txstrt => TXSTRT, txdlys => TXDLYS, cansfcs => CANSFCS, trxoff => TRXOFF, wait4resp => WAIT4RESP,
        rxenab => RXENAB, rxdlye => RXDLYE,
        hrbpt => HRBPT
    };
reg(encode, sys_ctrl, Val) ->
    #{
        sfcst := SFCST, txstrt := TXSTRT, txdlys := TXDLYS, cansfcs := CANSFCS, trxoff := TRXOFF, wait4resp := WAIT4RESP,
        rxenab := RXENAB, rxdlye := RXDLYE,
        hrbpt := HRBPT
    } = Val,
    <<
```

```
        WAIT4RESP:1, TRXOFF:1, 2#0:2, CANSFCS:1, TXDLYS:1, TXSTRT:1, SFCST:1, %
            bits 7-0
        2#0:6, RXDLYE:1, RXENAB:1, % bits 15-8
        2#0:8, % bits 23-16
        2#0:7, HRBPT:1 % bits 31-24
    >>;
reg(decode, sys_mask, Resp) ->
    <<
        MTXFRS:1, MTXPHS:1, MTXPRS:1, MTXFRB:1, MAAT:1, MESYNCR:1, MCPLOCK:1,
            Reserved0:1, % bits 7-0
        MRXFCE:1, MRXFCG:1, MRXDFR:1, MRXPHE:1, MRXPHD:1, MLDEDON:1, MRXSFDD:1,
            MRXPRD:1, % bits 15-8
        MSLP2INIT:1, MGPIOIRQ:1, MRXPTO:1, MRXOVRR:1, Reserved1:1, MLDEERR:1,
            MRXRFTO:1, MRXRFSL:1, % bits 23-16
        Reserved2:2, MAFFREJ:1, MTXBERR:1, MHPDDWAR:1, MPLLHILO:1, MCPLLLL:1,
            MRFPLLLL:1 % bits 31-24
    >> = Resp,
    #{
        mtxfrs => MTXFRS, mtxphs => MTXPHS, mtxprs => MTXPRS, mtxfrb => MTXFRB,
            maat => MAAT, mesyncr => MESYNCR, mcplock => MCPLOCK, res0 =>
            Reserved0, % bits 7-0
        mrxfce => MRXFCE, mrxfcg => MRXFCG, mrxdfr => MRXDFR, mrxphe => MRXPHE,
            mrxphd => MRXPHD, mldeon => MLDEDON, mrxsfdd => MRXSFDD, mrxprd =>
            MRXPRD, % bits 15-8
        mslp2init => MSLP2INIT, mgpioirq => MGPIOIRQ, mrxpto => MRXPTO, mrxovrr =>
             MRXOVRR, res1 => Reserved1, mldeerr => MLDEERR, mrxrfto => MRXRFTO,
            mrxrfsl => MRXRFSL, % bits 23-16
        res2 => Reserved2, maffrej => MAFFREJ, mtxberr => MTXBERR, mhpddwar =>
            MHPDDWAR, mpllhilo => MPLLHILO, mcplll => MCPLLLL, mrfpllll =>
            MRFPLLLL % bits 31-24
    };
reg(encode, sys_mask, Val) ->
    #{
        mtxfrs := MTXFRS, mtxphs := MTXPHS, mtxprs := MTXPRS, mtxfrb := MTXFRB,
            maat := MAAT, mesyncr := MESYNCR, mcplock := MCPLOCK, res0 :=
            Reserved0, % bits 7-0
        mrxfce := MRXFCE, mrxfcg := MRXFCG, mrxdfr := MRXDFR, mrxphe := MRXPHE,
            mrxphd := MRXPHD, mldeon := MLDEDON, mrxsfdd := MRXSFDD, mrxprd :=
            MRXPRD, % bits 15-8
        mslp2init := MSLP2INIT, mgpioirq := MGPIOIRQ, mrxpto := MRXPTO, mrxovrr :=
             MRXOVRR, res1 := Reserved1, mldeerr := MLDEERR, mrxrfto := MRXRFTO,
            mrxrfsl := MRXRFSL, % bits 23-16
        res2 := Reserved2, maffrej := MAFFREJ, mtxberr := MTXBERR, mhpddwar :=
            MHPDDWAR, mpllhilo := MPLLHILO, mcplll := MCPLLLL, mrfpllll :=
            MRFPLLLL % bits 31-24
    } = Val,
    <<
        MTXFRS:1, MTXPHS:1, MTXPRS:1, MTXFRB:1, MAAT:1, MESYNCR:1, MCPLOCK:1,
            Reserved0:1, % bits 7-0
        MRXFCE:1, MRXFCG:1, MRXDFR:1, MRXPHE:1, MRXPHD:1, MLDEDON:1, MRXSFDD:1,
            MRXPRD:1, % bits 15-8
        MSLP2INIT:1, MGPIOIRQ:1, MRXPTO:1, MRXOVRR:1, Reserved1:1, MLDEERR:1,
            MRXRFTO:1, MRXRFSL:1, % bits 23-16
        Reserved2:2, MAFFREJ:1, MTXBERR:1, MHPDDWAR:1, MPLLHILO:1, MCPLLLL:1,
            MRFPLLLL:1 % bits 31-24
    >>;
reg(decode, sys_status, Resp) ->
    <<
        TXFRS:1, TXPHS:1, TXPRS:1, TXFRB:1, AAT:1, ESYNCR:1, CPLOCK:1, IRQS:1, %
            bits 7-0
        RXFCE:1, RXFCG:1, RXDFR:1, RXPHE:1, RXPHD:1, LDEDONE:1, RXSFDD:1, RXPRD:1,
             % bits 15-8
```

```
        SPL2INIT:1, GPIOIRQ:1, RXPTO:1, RXOVRR:1, Reserved0:1, LDEERR:1, RXRFTO:1, RXRFSL:1, % bits 23-16
        ICRBP:1, HSRBP:1, AFFREJ:1, TXBERR:1, HPDWARN:1, RXSFDTO:1, CLCKPLL_LL:1, RFPLL_LL:1, % bits 31-24
        Reserved1:5, TXPUTE:1, RXPREJ:1, RXRSCS:1 % bits 39-32
    >> = Resp,
    #{
        txfrs => TXFRS, txphs => TXPHS, txprs => TXPRS, txfrb => TXFRB, aat => AAT, esyncr => ESYNCR, cplock => CPLOCK, irqs => IRQS, % bits 7-0
        rxfce => RXFCE, rxfcg => RXFCG, rxdfr => RXDFR, rxphe => RXPHE, rxphd => RXPHD, ldedone => LDEDONE, rxsfdd => RXSFDD, rxprd => RXPRD, % bits 15-8
        splt2init => SPL2INIT, gpioirq => GPIOIRQ, rxpto => RXPTO, rxovrr => RXOVRR, res0 => Reserved0, ldeerr => LDEERR, rxrfto => RXRFTO, rxrfsl => RXRFSL, % bits 23-16
        icrbp => ICRBP, hsrbp => HSRBP, affrej => AFFREJ, txberr => TXBERR, hdpwarn => HPDWARN, rxsfdto => RXSFDTO, clkpll_ll => CLCKPLL_LL, rfpll_ll => RFPLL_LL, % bits 31-24
        res1 => Reserved1, txpute => TXPUTE, rxprej => RXPREJ, rxrscs => RXRSCS
    };
reg(encode, sys_status, Val) ->
    #{
        txfrs := TXFRS, txphs := TXPHS, txprs := TXPRS, txfrb := TXFRB, aat := AAT, esyncr := ESYNCR, cplock := CPLOCK, irqs := IRQS, % bits 7-0
        rxfce := RXFCE, rxfcg := RXFCG, rxdfr := RXDFR, rxphe := RXPHE, rxphd := RXPHD, ldedone := LDEDONE, rxsfdd := RXSFDD, rxprd := RXPRD, % bits 15-8
        splt2init := SPL2INIT, gpioirq := GPIOIRQ, rxpto := RXPTO, rxovrr := RXOVRR, res0 := Reserved0, ldeerr := LDEERR, rxrfto := RXRFTO, rxrfsl := RXRFSL, % bits 23-16
        icrbp := ICRBP, hsrbp := HSRBP, affrej := AFFREJ, txberr := TXBERR, hdpwarn := HPDWARN, rxsfdto := RXSFDTO, clkpll_ll := CLCKPLL_LL, rfpll_ll := RFPLL_LL, % bits 31-24
        res1 := Reserved1, txpute := TXPUTE, rxprej := RXPREJ, rxrscs := RXRSCS
    } = Val,
    <<
        TXFRS:1, TXPHS:1, TXPRS:1, TXFRB:1, AAT:1, ESYNCR:1, CPLOCK:1, IRQS:1, % bits 7-0
        RXFCE:1, RXFCG:1, RXDFR:1, RXPHE:1, RXPHD:1, LDEDONE:1, RXSFDD:1, RXPRD:1, % bits 15-8
        SPL2INIT:1, GPIOIRQ:1, RXPTO:1, RXOVRR:1, Reserved0:1, LDEERR:1, RXRFTO:1, RXRFSL:1, % bits 23-16
        ICRBP:1, HSRBP:1, AFFREJ:1, TXBERR:1, HPDWARN:1, RXSFDTO:1, CLCKPLL_LL:1, RFPLL_LL:1, % bits 31-24
        Reserved1:5, TXPUTE:1, RXPREJ:1, RXRSCS:1 % bits 39-32
    >>;
reg(decode, rx_finfo, Resp) ->
    <<
        RXPACC:12, RXPSR:2, RXPRFR:2, RNG:1, RXBR:2, RXNSPL:2, _:1, RXFLE:3, RXFLEN:7
    >> = reverse(Resp),
    #{
        rxpacc => RXPACC, rxpsr => RXPSR, rxprfr => RXPRFR, rng => RNG, rxbr => RXBR, rxnspl => RXNSPL, rxfle => RXFLE, rxflen => RXFLEN
    };
reg(decode, rx_buffer, Resp) ->
    #{ rx_buffer => reverse(Resp)};
reg(decode, rx_fqual, Resp) ->
    <<
        CIR_PWR:16, PP_AMPL3:16, FP_AMPL2:16, STD_NOISE:16
    >> = Resp,
    #{
```

```
        cir_pwr => CIR_PWR, pp_ampl3 => PP_AMPL3, fp_ampl2 => FP_AMPL2, std_noise
            => STD_NOISE
    };
reg(decode, rx_ttcki, Resp) ->
    <<
      RXTTCKI:32
    >> = reverse(Resp),
    #{
        rxttcki => RXTTCKI
    };
reg(decode, rx_ttcko, Resp) ->
    <<
        _:1, RCPHASE:7, RSMPDEL:8, _:5, RXTOFS:19
    >> = reverse(Resp),
    #{
        rcphase => RCPHASE, rsmpdel => RSMPDEL, rxtofs => RXTOFS
    };
reg(decode, rx_time, Resp) ->
    <<
        RX_RAWST:40, FP_AMPL1:16, FP_INDEX:16, RX_STAMP:40
    >> = reverse(Resp),
    #{
        rx_rawst => RX_RAWST, fp_ampl1 => FP_AMPL1, fp_index => FP_INDEX, rx_stamp
             => RX_STAMP
    };
reg(decode, tx_time, Resp) ->
    <<
        TX_RAWST:40, TX_STAMP:40
    >> = reverse(Resp),
    #{
        tx_rawst => TX_RAWST, tx_stamp => TX_STAMP
    };
reg(decode, tx_antd, Resp) ->
    #{
        tx_antd => reverse(Resp)
    };
reg(encode, tx_antd, Val) ->
    #{
        tx_antd := TX_ANTD
    } = Val,
    reverse(<<
        TX_ANTD:16
    >>);
reg(decode, sys_state, Resp) ->
    <<
        _:8, _:4,  PMSC_STATE:4, _:3, RX_STATE:5, _:4, TX_STATE:4
    >> = reverse(Resp),
    #{
        pmsc_state => PMSC_STATE, rx_state => RX_STATE, tx_state => TX_STATE
    };
reg(decode, ack_resp_t, Resp) ->
    <<
        ACK_TIME:8, _:4, W4R_TIME:20
    >> = reverse(Resp),
    #{
        ack_tim => ACK_TIME, w4r_tim => W4R_TIME
    };
reg(encode, ack_resp_t, Val) ->
    #{
        ack_tim := ACK_TIME, w4r_tim := W4R_TIME
    } = Val,
    reverse(<<
```

```
        ACK_TIME:8, 2#0:4, W4R_TIME:20
    >>);
reg(decode, rx_sniff, Resp) ->
    <<
        Reserved0:16, SNIFF_OFFT:8, Reserved1:4, SNIFF_ONT:4
    >> = reverse(Resp),
    #{
        res0 => Reserved0,
        sniff_offt => SNIFF_OFFT,
        sniff_ont => SNIFF_ONT,
        res1 => Reserved1
    };
reg(encode, rx_sniff, Val) ->
    #{
        res0 := Reserved0,
        sniff_offt := SNIFF_OFFT,
        sniff_ont := SNIFF_ONT,
        res1 := Reserved1
    } = Val,
    reverse(<<
        Reserved0:16, SNIFF_OFFT:8, Reserved1:4, SNIFF_ONT:4
    >>);
% Smart transmit power control (cf. user manual p 104)
reg(decode, tx_power, Resp) ->
    <<
        BOOSTP125:8, BOOSTP250:8, BOOSTP500:8, BOOSTNORM:8
    >> = reverse(Resp),
    #{
        boostp125 => BOOSTP125, boostp250 => BOOSTP250, boostp500 => BOOSTP500,
            boostnorm => BOOSTNORM
    };
reg(encode, tx_power, Val) ->
    % Leave the possibility to write the value as one
    case Val of
        #{ tx_power := ValToEncode } -> reverse(<<ValToEncode:32>>);
        #{ boostp125 := BOOSTP125, boostp250 := BOOSTP250, boostp500 := BOOSTP500,
             boostnorm := BOOSTNORM } ->reverse(<<BOOSTP125:8, BOOSTP250:8,
            BOOSTP500:8, BOOSTNORM:8>>)
    end;
reg(decode, chan_ctrl, Resp) ->
    <<
        RX_PCODE:5, TX_PCODE:5, RNSSFD:1, TNSSFD:1, RXPRF:2, DWSFD:1, Reserved0:9,
             RX_CHAN:4, TX_CHAN:4
    >> = reverse(Resp),
    #{
        rx_pcode => RX_PCODE, tx_pcode => TX_PCODE, rnssfd => RNSSFD, tnssfd =>
            TNSSFD, rxprf => RXPRF, dwsfd => DWSFD, res0 => Reserved0, rx_chan =>
            RX_CHAN, tx_chan => TX_CHAN
    };
reg(encode, chan_ctrl, Val) ->
    #{
        rx_pcode := RX_PCODE, tx_pcode := TX_PCODE, rnssfd := RNSSFD, tnssfd :=
            TNSSFD, rxprf := RXPRF, dwsfd := DWSFD, res0 := Reserved0, rx_chan :=
            RX_CHAN, tx_chan := TX_CHAN
    } = Val,
    reverse(<<
        RX_PCODE:5, TX_PCODE:5, RNSSFD:1, TNSSFD:1, RXPRF:2, DWSFD:1, Reserved0:9,
             RX_CHAN:4, TX_CHAN:4
    >>);
reg(encode, usr_sfd, Value) ->
    #{
      usr_sfd := USR_SFD
```

```
     } = Value,
    reverse(<<
        USR_SFD:(8*41)
    >>);
reg(decode, usr_sfd, Resp) ->
    <<
        USR_SFD:(8*41)
    >> = reverse(Resp),
    #{
      usr_sfd => USR_SFD
     };
% AGC_CTRL is a complex register with reserved bits that can't be written
reg(encode, agc_ctrl1, Val) ->
    #{
        res := Reserved, dis_am := DIS_AM
    } = Val,
    reverse(<<
        Reserved:15, DIS_AM:1
    >>);
reg(encode, agc_tune1, Val) ->
    #{
      agc_tune1 := AGC_TUNE1
     } = Val,
    reverse(<<
        AGC_TUNE1:16
    >>);
reg(encode, agc_tune2, Val) ->
    #{
        agc_tune2 := AGC_TUNE2
     } = Val,
    reverse(<<
        AGC_TUNE2:32
    >>);
reg(encode, agc_tune3, Val) ->
    #{
        agc_tune3 := AGC_TUNE3
     } = Val,
    reverse(<<
        AGC_TUNE3:16
    >>);
reg(decode, agc_ctrl, Resp) ->
    <<
        _:4, EDV2:9, EDG1:5, _:6, % AGC_STAT1 (RP => don't save reserved bits)
        _:80, % Reserved 4
        AGC_TUNE3:16, % AGC_TUNE3
        _:16, % Reserved 3
        AGC_TUNE2:32, % AGC_TUNE2
        _:48, % Reserved 2
        AGC_TUNE1:16, % AGC_TUNE1
        Reserved0:15, DIS_AM:1, % AGC_CTRL1 (RW => save reserved bits)
        _:16 % Reserved 1
    >> = reverse(Resp),
    #{
        agc_ctrl1 => #{res => Reserved0, dis_am => DIS_AM},
        agc_tune1 => AGC_TUNE1,
        agc_tune2 => AGC_TUNE2,
        agc_tune3 => AGC_TUNE3,
        agc_stat1 => #{edv2 => EDV2, edg1 => EDG1}
    };
reg(encode, ec_ctrl, Val) ->
    #{
       res := Reserved, ostrm := OSTRM, wait := WAIT, pllldt := PLLLDT, osrsm :=
```

```
            OSRSM, ostsm := OSTSM
      } = Val,
     reverse(<<
         Reserved:20, OSTRM:1, WAIT:8, PLLLDT:1, OSRSM:1, OSTSM:1 % EC_CTRL
     >>);
reg(decode, ext_sync, Resp) ->
     <<
         _:26, OFFSET_EXT:6, % EC_GLOP
         RX_TS_EST:32, % EC_RXTC
         Reserved:20, OSTRM:1, WAIT:8, PLLLDT:1, OSRSM:1, OSTSM:1 % EC_CTRL
     >> = reverse(Resp),
     #{
         ec_ctrl => #{res => Reserved, ostrm => OSTRM, wait => WAIT, pllldt =>
             PLLLDT, osrsm => OSRSM, ostsm => OSTSM},
         rx_ts_est => RX_TS_EST,
         ec_golp => #{offset_ext => OFFSET_EXT}
     };
% "The host system doesn't need to access the ACC_MEM in normal operation, however
      it may be of interest [...] for diagnostic purpose" (from DW1000 user manual)
reg(decode, acc_mem, Resp) ->
     #{
         acc_mem => reverse(Resp)
     };
reg(encode, gpio_mode, Val) ->
     #{
       msgp8 := MSGP8, msgp7 := MSGP7, msgp6 := MSGP6, msgp5 := MSGP5, msgp4 :=
           MSGP4, msgp3 := MSGP3, msgp2 := MSGP2, msgp1 := MSGP1, msgp0 := MSGP0
      } = Val,
     reverse(<<
         2#0:8, MSGP8:2, MSGP7:2, MSGP6:2, MSGP5:2, MSGP4:2, MSGP3:2, MSGP2:2,
             MSGP1:2, MSGP0:2, 2#0:6 % GPIO_MODE
     >>);
reg(encode, gpio_dir, Val) ->
     #{
         gdm8 := GDM8, gdm7 := GDM7, gdm6 := GDM6, gdm5 := GDM5, gdm4 := GDM4, gdm3
              := GDM3, gdm2 := GDM2, gdm1 := GDM1, gdm0 := GDM0,
         gdp8 := GDP8, gdp7 := GDP7, gdp6 := GDP6, gdp5 := GDP5, gdp4 := GDP4, gdp3
              := GDP3, gdp2 := GDP2, gdp1 := GDP1, gdp0 := GDP0
      } = Val,
     reverse(<<
         2#0:11, GDM8:1, 2#0:3, GDP8:1, GDM7:1, GDM6:1, GDM5:1, GDM4:1, GDP7:1,
             GDP6:1, GDP5:1, GDP4:1, GDM3:1, GDM2:1, GDM1:1, GDM0:1, GDP3:1, GDP2
             :1, GDP1:1, GDP0:1 % GPIO2_DIR
     >>);
reg(encode, gpio_dout, Val) ->
     #{
         gom8 := GOM8, gom7 := GOM7, gom6 := GOM6, gom5 := GOM5, gom4 := GOM4, gom3
              := GOM3, gom2 := GOM2, gom1 := GOM1, gom0 := GOM0,
         gop8 := GOP8, gop7 := GOP7, gop6 := GOP6, gop5 := GOP5, gop4 := GOP4, gop3
              := GOP3, gop2 := GOP2, gop1 := GOP1, gop0 := GOP0
      } = Val,
     reverse(<<
         2#0:11, GOM8:1, 2#0:3, GOP8:1, GOM7:1, GOM6:1, GOM5:1, GOM4:1, GOP7:1,
             GOP6:1, GOP5:1, GOP4:1, GOM3:1, GOM2:1, GOM1:1, GOM0:1, GOP3:1, GOP2
             :1, GOP1:1, GOP0:1 % GPIO_DOUT
     >>);
reg(encode, gpio_irqe, Val) ->
     #{
         girqe8 := GIRQE8, girqe7 := GIRQE7, girqe6 := GIRQE6, girqe5 := GIRQE5,
             girqe4 := GIRQE4, girqe3 := GIRQE3, girqe2 := GIRQE2, girqe1 := GIRQE1
             , girqe0 := GIRQE0
      } = Val,
```

```
    reverse(<<
        2#0:23, GIRQE8:1, GIRQE7:1, GIRQE6:1, GIRQE5:1, GIRQE4:1, GIRQE3:1, GIRQE2:1, GIRQE1:1, GIRQE0:1 % GPIO_IRQE
    >>);
reg(encode, gpio_isen, Val) ->
    #{
        gisen8 := GISEN8, gisen7 := GISEN7, gisen6 := GISEN6, gisen5 := GISEN5, gisen4 := GISEN4, gisen3 := GISEN3, gisen2 := GISEN2, gisen1 := GISEN1, gisen0 := GISEN0
     } = Val,
    reverse(<<
        2#0:23, GISEN8:1, GISEN7:1, GISEN6:1, GISEN5:1, GISEN4:1, GISEN3:1, GISEN2:1, GISEN1:1, GISEN0:1 % GPIO_ISEN
    >>);
reg(encode, gpio_imod, Val) ->
    #{
        gimod8 := GIMOD8, gimod7 := GIMOD7, gimod6 := GIMOD6, gimod5 := GIMOD5, gimod4 := GIMOD4, gimod3 := GIMOD3, gimod2 := GIMOD2, gimod1 := GIMOD1, gimod0 := GIMOD0
     } = Val,
    reverse(<<
        2#0:23, GIMOD8:1, GIMOD7:1, GIMOD6:1, GIMOD5:1, GIMOD4:1, GIMOD3:1, GIMOD2:1, GIMOD1:1, GIMOD0:1 % GPIO_IMOD
    >>);
reg(encode, gpio_ibes, Val) ->
    #{
        gibes8 := GIBES8, gibes7 := GIBES7, gibes6 := GIBES6, gibes5 := GIBES5, gibes4 := GIBES4, gibes3 := GIBES3, gibes2 := GIBES2, gibes1 := GIBES1, gibes0 := GIBES0
     } = Val,
    reverse(<<
        2#0:23, GIBES8:1, GIBES7:1, GIBES6:1, GIBES5:1, GIBES4:1, GIBES3:1, GIBES2:1, GIBES1:1, GIBES0:1 % GPIO_IBES
    >>);
reg(encode, gpio_iclr, Val) ->
    #{
        giclr8 := GICLR8, giclr7 := GICLR7, giclr6 := GICLR6, giclr5 := GICLR5, giclr4 := GICLR4, giclr3 := GICLR3, giclr2 := GICLR2, giclr1 := GICLR1, giclr0 := GICLR0
     } = Val,
    reverse(<<
        2#0:23, GICLR8:1, GICLR7:1, GICLR6:1, GICLR5:1, GICLR4:1, GICLR3:1, GICLR2:1, GICLR1:1, GICLR0:1 % GPIO_ICLR
    >>);
reg(encode, gpio_idbe, Val) ->
    #{
        gidbe8 := GIDBE8, gidbe7 := GIDBE7, gidbe6 := GIDBE6, gidbe5 := GIDBE5, gidbe4 := GIDBE4, gidbe3 := GIDBE3, gidbe2 := GIDBE2, gidbe1 := GIDBE1, gidbe0 := GIDBE0
     } = Val,
    reverse(<<
        2#0:23, GIDBE8:1, GIDBE7:1, GIDBE6:1, GIDBE5:1, GIDBE4:1, GIDBE3:1, GIDBE2:1, GIDBE1:1, GIDBE0:1 % GPIO_IDBE
    >>);
reg(encode, gpio_raw, Val) ->
    #{
        grawp8 := GRAWP8, grawp7 := GRAWP7, grawp6 := GRAWP6, grawp5 := GRAWP5, grawp4 := GRAWP4, grawp3 := GRAWP3, grawp2 := GRAWP2, grawp1 := GRAWP1, grawp0 := GRAWP0
     } = Val,
    reverse(<<
        2#0:23, GRAWP8:1, GRAWP7:1, GRAWP6:1, GRAWP5:1, GRAWP4:1, GRAWP3:1, GRAWP2
```

```
            :1, GRAWP1:1, GRAWP0:1 % GPIO_RAW
    >>);
reg(decode, gpio_ctrl, Resp) ->
    <<
        _:23, GRAWP8:1, GRAWP7:1, GRAWP6:1, GRAWP5:1, GRAWP4:1, GRAWP3:1, GRAWP2
            :1, GRAWP1:1, GRAWP0:1, % GPIO_RAW
        _:23, GIDBE8:1, GIDBE7:1, GIDBE6:1, GIDBE5:1, GIDBE4:1, GIDBE3:1, GIDBE2
            :1, GIDBE1:1, GIDBE0:1, % GPIO_IDBE
        _:23, GICLR8:1, GICLR7:1, GICLR6:1, GICLR5:1, GICLR4:1, GICLR3:1, GICLR2
            :1, GICLR1:1, GICLR0:1, % GPIO_ICLR
        _:23, GIBES8:1, GIBES7:1, GIBES6:1, GIBES5:1, GIBES4:1, GIBES3:1, GIBES2
            :1, GIBES1:1, GIBES0:1, % GPIO_IBES
        _:23, GIMOD8:1, GIMOD7:1, GIMOD6:1, GIMOD5:1, GIMOD4:1, GIMOD3:1, GIMOD2
            :1, GIMOD1:1, GIMOD0:1, % GPIO_IMOD
        _:23, GISEN8:1, GISEN7:1, GISEN6:1, GISEN5:1, GISEN4:1, GISEN3:1, GISEN2
            :1, GISEN1:1, GISEN0:1, % GPIO_ISEN
        _:23, GIRQE8:1, GIRQE7:1, GIRQE6:1, GIRQE5:1, GIRQE4:1, GIRQE3:1, GIRQE2
            :1, GIRQE1:1, GIRQE0:1, % GPIO_IRQE
        _:11, GOM8:1, _:3, GOP8:1, GOM7:1, GOM6:1, GOM5:1, GOM4:1, GOP7:1, GOP6:1,
             GOP5:1, GOP4:1, GOM3:1, GOM2:1, GOM1:1, GOM0:1, GOP3:1, GOP2:1, GOP1
            :1, GOP0:1, % GPIO_DOUT
        _:11, GDM8:1, _:3, GDP8:1, GDM7:1, GDM6:1, GDM5:1, GDM4:1, GDP7:1, GDP6:1,
             GDP5:1, GDP4:1, GDM3:1, GDM2:1, GDM1:1, GDM0:1, GDP3:1, GDP2:1, GDP1
            :1, GDP0:1, % GPIO_DIR
        _:32, % Reserved
        _:8, MSGP8:2, MSGP7:2, MSGP6:2, MSGP5:2, MSGP4:2, MSGP3:2, MSGP2:2, MSGP1
            :2, MSGP0:2, _:6 % GPIO_MODE
    >> = reverse(Resp),
    #{
        gpio_mode => #{msgp8 => MSGP8, msgp7 => MSGP7, msgp6 => MSGP6, msgp5 =>
            MSGP5, msgp4 => MSGP4, msgp3 => MSGP3, msgp2 => MSGP2, msgp1 => MSGP1,
             msgp0 => MSGP0},
        gpio_dir => #{gdm8 => GDM8, gdm7 => GDM7, gdm6 => GDM6, gdm5 => GDM5, gdm4
             => GDM4, gdm3 => GDM3, gdm2 => GDM2, gdm1 => GDM1, gdm0 => GDM0,
                      gdp8 => GDP8, gdp7 => GDP7, gdp6 => GDP6, gdp5 => GDP5, gdp4
                          => GDP4, gdp3 => GDP3, gdp2 => GDP2, gdp1 => GDP1, gdp0
                          => GDP0},
        gpio_dout => #{gom8 => GOM8, gom7 => GOM7, gom6 => GOM6, gom5 => GOM5,
            gom4 => GOM4, gom3 => GOM3, gom2 => GOM2, gom1 => GOM1, gom0 => GOM0,
                       gop8 => GOP8, gop7 => GOP7, gop6 => GOP6, gop5 => GOP5,
                           gop4 => GOP4, gop3 => GOP3, gop2 => GOP2, gop1 => GOP1,
                            gop0 => GOP0},
        gpio_irqe => #{girqe8 => GIRQE8, girqe7 => GIRQE7, girqe6 => GIRQE6,
            girqe5 => GIRQE5, girqe4 => GIRQE4, girqe3 => GIRQE3, girqe2 => GIRQE2
            , girqe1 => GIRQE1, girqe0 => GIRQE0},
        gpio_isen => #{gisen8 => GISEN8, gisen7 => GISEN7, gisen6 => GISEN6,
            gisen5 => GISEN5, gisen4 => GISEN4, gisen3 => GISEN3, gisen2 => GISEN2
            , gisen1 => GISEN1, gisen0 => GISEN0},
        gpio_imod => #{gimod8 => GIMOD8, gimod7 => GIMOD7, gimod6 => GIMOD6,
            gimod5 => GIMOD5, gimod4 => GIMOD4, gimod3 => GIMOD3, gimod2 => GIMOD2
            , gimod1 => GIMOD1, gimod0 => GIMOD0},
        gpio_ibes => #{gibes8 => GIBES8, gibes7 => GIBES7, gibes6 => GIBES6,
            gibes5 => GIBES5, gibes4 => GIBES4, gibes3 => GIBES3, gibes2 => GIBES2
            , gibes1 => GIBES1, gibes0 => GIBES0},
        gpio_iclr => #{giclr8 => GICLR8, giclr7 => GICLR7, giclr6 => GICLR6,
            giclr5 => GICLR5, giclr4 => GICLR4, giclr3 => GICLR3, giclr2 => GICLR2
            , giclr1 => GICLR1, giclr0 => GICLR0},
        gpio_idbe => #{gidbe8 => GIDBE8, gidbe7 => GIDBE7, gidbe6 => GIDBE6,
            gidbe5 => GIDBE5, gidbe4 => GIDBE4, gidbe3 => GIDBE3, gidbe2 => GIDBE2
            , gidbe1 => GIDBE1, gidbe0 => GIDBE0},
        gpio_raw => #{grawp8 => GRAWP8, grawp7 => GRAWP7, grawp6 => GRAWP6, grawp5
             => GRAWP5, grawp4 => GRAWP4, grawp3 => GRAWP3, grawp2 => GRAWP2,
```

```
            grawp1 => GRAWP1, grawp0 => GRAWP0}
    };
reg(encode, drx_tune0b, Val) ->
    #{
        drx_tune0b := DRX_TUNE0b
     } = Val,
    reverse(<<
        DRX_TUNE0b:16
    >>);
reg(encode, drx_tune1a, Val) ->
    #{
        drx_tune1a := DRX_TUNE1a
     } = Val,
    reverse(<<
        DRX_TUNE1a:16
    >>);
reg(encode, drx_tune1b, Val) ->
    #{
        drx_tune1b := DRX_TUNE1b
     } = Val,
    reverse(<<
        DRX_TUNE1b:16
    >>);
reg(encode, drx_tune2, Val) ->
    #{
        drx_tune2 := DRX_TUNE2
     } = Val,
    reverse(<<
        DRX_TUNE2:32
    >>);
reg(encode, drx_sfdtoc, Val) ->
    #{
        drx_sfdtoc := DRX_SFDTOC
     } = Val,
    reverse(<<
        DRX_SFDTOC:16
    >>);
reg(encode, drx_pretoc, Val) ->
    #{
        drx_pretoc := DRX_PRETOC
     } = Val,
    reverse(<<
        DRX_PRETOC:16
    >>);
reg(encode, drx_tune4h, Val) ->
    #{
        drx_tune4h := DRX_TUNE4H
     } = Val,
    reverse(<<
        DRX_TUNE4H:16
    >>);
reg(decode, drx_conf, Resp) ->
    <<
        RXPACC_NOSAT:8, % present in the user manual but not in the driver code in
             C
        % _:8, % Placeholder for the remaining 8 bits
        DRX_CAR_INT:24,
        DRX_TUNE4H:16,
        DRX_PRETOC:16,
        _:16,
        DRX_SFDTOC:16,
        _:160,
```

```
        DRX_TUNE2:32,
        DRX_TUNE1b:16,
        DRX_TUNE1a:16,
        DRX_TUNE0b:16,
        _:16
    >> = reverse(Resp),
    #{
        drx_tune0b => DRX_TUNE0b,
        drx_tune1a => DRX_TUNE1a,
        drx_tune1b => DRX_TUNE1b,
        drx_tune2 => DRX_TUNE2,
        drx_tune4h => DRX_TUNE4H,
        drx_car_int => DRX_CAR_INT,
        drx_sfdtoc => DRX_SFDTOC,
        drx_pretoc => DRX_PRETOC,
        rxpacc_nosat => RXPACC_NOSAT
    };
reg(encode, rf_conf, Val) ->
    #{
        txrxsw := TXRXSW, ldofen := LDOFEN, pllfen := PLLFEN, txfen := TXFEN
     } = Val,
    reverse(<<
        2#0:9, TXRXSW:2, LDOFEN:5, PLLFEN:3, TXFEN:5, 2#0:8 % RF_CONF
    >>);
reg(encode, rf_rxctrlh, Val) ->
    #{
        rf_rxctrlh := RF_RXCTRLH
     } = Val,
    reverse(<<
        RF_RXCTRLH:8 % RF_RXCTRLH
    >>);
% user manual gives fields but encoding should be done as one following table 38
reg(encode, rf_txctrl, Val) ->
    #{
      rf_txctrl := RF_TXCTRL
     } = Val,
    reverse(<<
        RF_TXCTRL:32
    >>);
reg(encode, ldotune, Val) ->
    #{
      ldotune := LDOTUNE
     } = Val,
    reverse(<<
        LDOTUNE:40
    >>);
reg(decode, rf_conf, Resp) ->
    <<
        _:40, % Placeholder for the remaining 40 bits
        LDOTUNE:40, % LDOTUNE
        _:28, RFPLLLOCK:1, CPLLHIGH:1, CPLLLOW:1, CPLLLOCK:1, % RF_STATUS
        _:128, _:96, % Reserved 2 - On user manual 16 bytes but offset gives 28 bytes (16 bytes (128 bits) + 12 bytes (96 bits))
        RF_TXCTRL:32, % cf. encode function: Reserved:20, TXMQ:3, TXMTUNE:4, _:5 - RF_TXCTRL
        RF_RXCTRLH:8, % RF_RXCTRLH
        _:56, % Reserved 1
        _:9, TXRXSW:2, LDOFEN:5, PLLFEN:3, TXFEN:5, _:8 % RF_CONF
    >> = reverse(Resp),
    #{
        ldotune => LDOTUNE,
        rf_status => #{rfplllock => RFPLLLOCK, cplllow => CPLLLOW, cpllhigh =>
```

```
            CPLLHIGH, cplllock => CPLLLOCK},
        rf_txctrl => RF_TXCTRL,
        rf_rxctrlh => RF_RXCTRLH,
        rf_conf => #{txrxsw => TXRXSW, ldofen => LDOFEN, pllfen => PLLFEN, txfen => TXFEN}
    };
reg(encode, tc_sarc, Val) ->
    #{
        sar_ctrl := SAR_CTRL
     } = Val,
    reverse(<<
      2#0:15, SAR_CTRL:1
    >>);
reg(encode, tc_pg_ctrl, Val) ->
    #{
        pg_tmeas := PG_TMEAS, res := Reserved, pg_start := PG_START
     } = Val,
    reverse(<<
        2#0:2, PG_TMEAS:4, Reserved:1, PG_START:1
    >>);
reg(encode, tc_pgdelay, Val) ->
    #{
        tc_pgdelay := TC_PGDELAY
     } = Val,
    reverse(<<
        TC_PGDELAY:8
    >>);
reg(encode, tc_pgtest, Val) ->
    #{
        tc_pgtest := TC_PGTEST
     } = Val,
    reverse(<<
        TC_PGTEST:8
    >>);
reg(decode, tx_cal, Resp) ->
    <<
        TC_PGTEST:8, % TC_PGTEST
        TC_PGDELAY:8, % TC_PGDELAY
        _:4, DELAY_CNT:12, % TC_PG_STATUS
        _:2, PG_TMEAS:4, Reserved0:1, PG_START:1, % TC_PG_CTRL
        SAR_WTEMP:8, SAR_WVBAT:8, % TC_SARW
        _:8, SAR_LTEMP:8, SAR_LVBAT:8, % TC_SARL
        _:8, % Place holder to fill the gap between the offsets
        _:15, SAR_CTRL:1 % TC_SARC
    >> = reverse(Resp),
    #{
        tc_pgtest => TC_PGTEST,
        tc_pgdelay => TC_PGDELAY,
        tc_pg_status => #{delay_cnt => DELAY_CNT},
        tc_pg_ctrl => #{pg_tmeas => PG_TMEAS, res => Reserved0, pg_start => PG_START},
        tc_sarw => #{sar_wtemp => SAR_WTEMP, sar_wvbat => SAR_WVBAT},
        tc_sarl => #{sar_ltemp => SAR_LTEMP, sar_lvbat => SAR_LVBAT},
        tc_sarc => #{sar_ctrl => SAR_CTRL}
    };
reg(encode, fs_pllcfg, Val) ->
    #{
        fs_pllcfg := FS_PLLCFG
     } = Val,
    reverse(<<
        FS_PLLCFG:32
    >>);
```

```
reg(encode, fs_plltune, Val) ->
    #{
        fs_plltune := FS_PLLTUNE
     } = Val,
    reverse(<<
        FS_PLLTUNE:8
    >>);
reg(encode, fs_xtalt, Val) ->
    #{
      res := Reserved, xtalt := XTALT
     } = Val,
    reverse(<<
        Reserved:3, XTALT:5
    >>);
reg(decode, fs_ctrl, Resp) ->
    <<
        _:48, % Reserved 3
        Reserved:3, XTALT:5, % FS_XTALT
        _:16, % Reserved 2
        FS_PLLTUNE:8, % FS_PLLTUNE
        FS_PLLCFG:32, % FS_PLLCFG
        _:56 % Reserved 1
    >> = reverse(Resp),
    #{
        fs_xtalt => #{res => Reserved, xtalt => XTALT},
        fs_plltune => FS_PLLTUNE,
        fs_pllcfg => FS_PLLCFG
    };
reg(encode, aon_wcfg, Val) ->
    #{
        onw_lld := ONW_LLD, onw_llde := ONW_LLDE, pres_slee := PRES_SLEE, own_l64
            := OWN_L64, own_ldc := OWN_LDC, own_leui := OWN_LEUI, own_rx := OWN_RX
            , own_rad := OWN_RAD
     } = Val,
    reverse(<<
        2#0:3, ONW_LLD:1, ONW_LLDE:1, 2#0:2, PRES_SLEE:1, OWN_L64:1, OWN_LDC:1,
            2#0:2, OWN_LEUI:1, 2#0:1, OWN_RX:1, OWN_RAD:1 % AON_WCFG
    >>);
reg(encode, aon_ctrl, Val) ->
    #{
        dca_enab := DCA_ENAB, dca_read := DCA_READ, upl_cfg := UPL_CFG, save :=
            SAVE, restore := RESTORE
     } = Val,
    reverse(<<
        DCA_ENAB:1, 2#0:3, DCA_READ:1, UPL_CFG:1, SAVE:1, RESTORE:1 % AON_CTRL
    >>);
reg(encode, aon_rdat, Val) ->
    #{
        aon_rdat := AON_RDAT
     } = Val,
    reverse(<<
        AON_RDAT:8 % AON_RDAT
    >>);
reg(encode, aon_addr, Val) ->
    #{
        aon_addr := AON_ADDR
     } = Val,
    reverse(<<
        AON_ADDR:8 % AON_ADDR
    >>);
reg(encode, aon_cfg0, Val) ->
    #{
```

```
        sleep_tim := SLEEP_TIM, lpclkdiva := LPCLKDIVA, lpdiv_en := LPDIV_EN,
            wake_cnt := WAKE_CNT, wake_spi := WAKE_SPI, wake_pin := WAKE_PIN,
            sleep_en := SLEEP_EN
     } = Val,
    reverse(<<
        SLEEP_TIM:16, LPCLKDIVA:11, LPDIV_EN:1, WAKE_CNT:1, WAKE_SPI:1, WAKE_PIN
            :1, SLEEP_EN:1 % AON_CFG0
    >>);
reg(encode, aon_cfg1, Val) ->
    #{
        res := Reserved, lposc_c := LPOSC_C, smxx := SMXX, sleep_ce := SLEEP_CE
     } = Val,
    reverse(<<
        Reserved:13, LPOSC_C:1, SMXX:1, SLEEP_CE:1 % AON_CFG1
    >>);
reg(decode, aon, Resp) ->
    <<
        Reserved:13, LPOSC_C:1, SMXX:1, SLEEP_CE:1, % AON_CFG1
        SLEEP_TIM:16, LPCLKDIVA:11, LPDIV_EN:1, WAKE_CNT:1, WAKE_SPI:1, WAKE_PIN
            :1, SLEEP_EN:1, % AON_CFG0
        _:8, % Reserved 1
        AON_ADDR:8, % AON_ADDR
        AON_RDAT:8, % AON_RDAT
        DCA_ENAB:1, _:3, DCA_READ:1, UPL_CFG:1, SAVE:1, RESTORE:1, % AON_CTRL
        _:3, ONW_LLD:1, ONW_LLDE:1, _:2, PRES_SLEE:1, OWN_L64:1, OWN_LDC:1, _:2,
            OWN_LEUI:1, _:1, OWN_RX:1, OWN_RAD:1 % AON_WCFG
    >> = reverse(Resp),
    #{
        aon_cfg1 => #{res => Reserved, lposc_c => LPOSC_C, smxx => SMXX, sleep_ce
            => SLEEP_CE},
        aon_cfg0 => #{sleep_tim => SLEEP_TIM, lpclkdiva => LPCLKDIVA, lpdiv_en =>
            LPDIV_EN, wake_cnt => WAKE_CNT, wake_spi => WAKE_SPI, wake_pin =>
            WAKE_PIN, sleep_en => SLEEP_EN},
        aon_addr => AON_ADDR,
        aon_rdat => AON_RDAT,
        aon_ctrl => #{dca_enab => DCA_ENAB, dca_read => DCA_READ, upl_cfg =>
            UPL_CFG, save => SAVE, restore => RESTORE},
        aon_wcfg => #{onw_lld => ONW_LLD, onw_llde => ONW_LLDE, pres_slee =>
            PRES_SLEE, own_l64 => OWN_L64, own_ldc => OWN_LDC, own_leui =>
            OWN_LEUI, own_rx => OWN_RX, own_rad => OWN_RAD}
    };
reg(encode, otp_wdat, Val) ->
    #{
        otp_wdat := OTP_WDAT
     } = Val,
    reverse(<<
        OTP_WDAT:32 % OTP_WDAT
    >>);
reg(encode, otp_addr, Val) ->
    #{
        otpaddr := OTP_ADDR, res := Reserved
     } = Val,
    reverse(<<
        Reserved:5, OTP_ADDR:11 % OTP_ADDR
    >>);
reg(encode, otp_ctrl, Val) ->
    #{
        ldeload := LDELOAD, res1 := Reserved1, otpmr := OTPMR, otpprog := OTPPROG,
             res2 := Reserved2, otpmrwr := OTPMRWR, res3 := Reserved3, otpread :=
            OTPREAD, otp_rden := OTPRDEN
     } = Val,
    reverse(<<
```

```
        LDELOAD:1, Reserved1:4, OTPMR:4, OTPPROG:1, Reserved2:2, OTPMRWR:1,
            Reserved3:1, OTPREAD:1, OTPRDEN:1 % OTP_CTRL
    >>);
reg(encode, otp_stat, Val) ->
    #{
        res := Reserved, otp_vpok := OTP_VPOK, otpprgd := OTPPRGD
     } = Val,
    reverse(<<
        Reserved:14, OTP_VPOK:1, OTPPRGD:1 % OTP_STAT
    >>);
reg(encode, otp_rdat, Val) ->
    #{
        otp_rdat := OTP_RDAT
     } = Val,
    reverse(<<
        OTP_RDAT:32 % OTP_RDAT
    >>);
reg(encode, opt_srdat, Val) ->
    #{
        otp_srdat := OTP_SRDAT
     } = Val,
    reverse(<<
        OTP_SRDAT:32 % OTP_SRDAT
    >>);
reg(encode, otp_sf, Val) ->
    #{
        res1 := Reserved1, ops_sel := OPS_SEL, res2 := Reserved2, ldo_kick :=
            LDO_KICK, ops_kick := OPS_KICK
     } = Val,
    reverse(<<
        Reserved1:2, OPS_SEL:1, Reserved2:3, LDO_KICK:1, OPS_KICK:1 % OTP_SF
    >>);
reg(decode, otp_if, Resp) ->
    <<
        Reserved5:2, OPS_SEL:1, Reserved6:3, LDO_KICK:1, OPS_KICK:1, % OTP_SF
        OTP_SRDAT:32, % OTP_SRDAT
        OTP_RDAT:32, % OTP_RDAT
        Reserved4:14, OTP_VPOK:1, OTPPRGD:1, % OTP_STAT
        LDELOAD:1, Reserved1:4, OTPMR:4, OTPPROG:1, Reserved2:2, OTPMRWR:1,
            Reserved3:1, OTPREAD:1, OTPRDEN:1, % OTP_CTRL
        Reserved0:5, OTP_ADDR:11, % OTP_ADDR
        OTP_WDAT:32 % OTP_WDAT
    >> = reverse(Resp),
    #{
        otp_sf => #{res1 => Reserved5, ops_sel => OPS_SEL, res2 => Reserved6,
            ldo_kick => LDO_KICK, ops_kick => OPS_KICK},
        otp_srdat => OTP_SRDAT,
        otp_rdat => OTP_RDAT,
        otp_stat => #{res => Reserved4, otp_vpok => OTP_VPOK, otpprgd => OTPPRGD},
        otp_ctrl => #{ldeload => LDELOAD, res1 => Reserved1, otpmr => OTPMR,
            otpprog => OTPPROG, res2 => Reserved2, otpmrwr => OTPMRWR, res3 =>
            Reserved3, otpread => OTPREAD, otp_rden => OTPRDEN},
        otp_addr => #{otpaddr => OTP_ADDR, res => Reserved0},
        otp_wdat => OTP_WDAT
    };
reg(decode, lde_thresh, Resp) ->
    <<
      LDE_THRESH:16
    >> = reverse(Resp),
    #{
      lde_thresh => LDE_THRESH
    };
```

```
reg(encode, lde_cfg1, Val) ->
    #{
      pmult := PMULT, ntm := NTM
     } = Val,
    reverse(<<
        PMULT:3, NTM:5
    >>);
reg(decode, lde_cfg1, Resp) ->
    <<
      PMULT:3, NTM:5
    >> = reverse(Resp),
    #{
      lde_cfg1 => #{pmult => PMULT, ntm => NTM}
    };
reg(decode, lde_ppindx, Resp) ->
    <<
      LDE_PPINDX:16
    >> = reverse(Resp),
    #{
      lde_ppindx => LDE_PPINDX
    };
reg(decode, lde_ppampl, Resp) ->
    <<
      LDE_PPAMPL:16
    >> = reverse(Resp),
    #{
      lde_ppampl => LDE_PPAMPL
    };
reg(encode, lde_rxantd, Val) ->
    #{
      lde_rxantd := LDE_RXANTD
     } = Val,
    reverse(<<
        LDE_RXANTD:16
    >>);
reg(decode, lde_rxantd, Resp) ->
    <<
      LDE_RXANTD:16
    >> = reverse(Resp),
    #{
      lde_rxantd => LDE_RXANTD
    };
reg(encode, lde_cfg2, Val) ->
    #{
        lde_cfg2 := LDE_CFG2
     } = Val,
    reverse(<<
        LDE_CFG2:16
    >>);
reg(decode, lde_cfg2, Resp) ->
    <<
      LDE_CFG2:16
    >> = reverse(Resp),
    #{
      lde_cfg2 => LDE_CFG2
    };
reg(encode, lde_repc, Val) ->
    #{
        lde_repc := LDE_REPC
     } = Val,
    reverse(<<
        LDE_REPC:16
```

```
    >>);
reg(decode, lde_repc, Resp) ->
    <<
      LDE_REPC:16
    >> = reverse(Resp),
    #{
      lde_repc => LDE_REPC
    };
reg(encode, evc_ctrl, Val) ->
    #{
        evc_clr := EVC_CLR, evc_en := EVC_EN
     } = Val,
    reverse(<<
        2#0:30, EVC_CLR:1, EVC_EN:1 % EVC_CTRL
    >>);
reg(encode, diag_tmc, Val) ->
    #{
        tx_pstm := TX_PSTM
     } = Val,
    reverse(<<
        2#0:11, TX_PSTM:1, 2#0:4 % DIAG_TMC
    >>);
reg(decode, dig_diag, Resp) ->
    <<
        _:11, TX_PSTM:1, _:4, % DIAG_TMC
        _:64, % Reserved 1
        _:4, EVC_TPW:12, % EVC_TPW
        _:4, EVC_HPW:12, % EVC_HPW
        _:4, EVC_TXFS:12, % EVC_TXFS
        _:4, EVC_FWTO:12, % EVC_FWTO
        _:4, EVC_PTO:12, % EVC_PTO
        _:4, EVC_STO:12, % EVC_STO
        _:4, ECV_OVR:12, % EVC_OVR
        _:4, EVC_FFR:12, % EVC_FFR
        _:4, EVC_FCE:12, % EVC_FCE
        _:4, EVC_FCG:12, % EVC_FCG
        _:4, EVC_RSE:12, % EVC_RSE
        _:4, EVC_PHE:12, % EVC_PHE
        _:30, EVC_CLR:1, EVC_EN:1 % EVC_CTRL
    >> = reverse(Resp),
    #{
        diag_tmc => #{tx_pstm => TX_PSTM},
        evc_tpw => EVC_TPW,
        evc_hpw => EVC_HPW,
        evc_txfs => EVC_TXFS,
        evc_fwto => EVC_FWTO,
        evc_pto => EVC_PTO,
        evc_sto => EVC_STO,
        evc_ovr => ECV_OVR,
        evc_ffr => EVC_FFR,
        evc_fce => EVC_FCE,
        evc_fcg => EVC_FCG,
        evc_rse => EVC_RSE,
        evc_phe => EVC_PHE,
        evc_ctrl => #{evc_clr => EVC_CLR, evc_en => EVC_EN}
    };
reg(encode, pmsc_ctrl0, Val) ->
    #{
        softreset := SOFTRESET, pll2_seq_en := PLL2_SEQ_EN, khzclken := KHZCLKEN,
            gpdrn := GPDRN, gpdce := GPDCE,
        gprn := GPRN, gpce := GPCE, amce := AMCE, adcce := ADCCE, otp := OTP, res8
             := Res8, res7 := Res7, face := FACE, txclks := TXCLKS, rxclks :=
```

```
            RXCLKS, sysclks := SYSCLKS % Here we need res8 for the initial config
            of the DW1000. We need to write it
     } = Val,
    reverse(<<
        SOFTRESET:4, 2#000:3, PLL2_SEQ_EN:1, KHZCLKEN:1, 2#011:3, GPDRN:1, GPDCE
            :1, GPRN:1, GPCE:1, AMCE:1, 2#0000:4, ADCCE:1, OTP:1, Res8:1, Res7:1,
            FACE:1, TXCLKS:2, RXCLKS:2, SYSCLKS:2 % PMSC_CTRL0
    >>);
reg(encode, pmsc_ctrl1, Val) ->
    #{
        khzclkdiv := KHZCLKDIV, lderune := LDERUNE, pllsyn := PLLSYN, snozr :=
            SNOZR, snoze := SNOZE, arxslp := ARXSLP, atxslp := ATXSLP, pktseq :=
            PKTSEQ, arx2init := ARX2INIT
     } = Val,
    reverse(<<
        KHZCLKDIV:6, 2#01000000:8, LDERUNE:1, 2#0:1, PLLSYN:1, SNOZR:1, SNOZE:1,
            ARXSLP:1, ATXSLP:1, PKTSEQ:8, 2#0:1, ARX2INIT:1, 2#0:1 % PMSC_CTRL1
    >>);
reg(encode, pmsc_snozt, Val) ->
    #{
        snoz_tim := SNOZ_TIM
     } = Val,
    reverse(<<
        SNOZ_TIM:8 % PMSC_SNOZT
    >>);
reg(encode, pmsc_txfseq, Val) ->
    #{
        txfineseq := TXFINESEQ
     } = Val,
    reverse(<<
        TXFINESEQ:16 % PMSC_TXFINESEQ
    >>);
reg(encode, pmsc_ledc, Val) ->
    #{
        res31 := RES31, blnknow := BLNKNOW, res15 := RES15, blnken := BLNKEN,
            blink_tim := BLINK_TIM
     } = Val,
    reverse(<<
        RES31:12, BLNKNOW:4, RES15:7, BLNKEN:1, BLINK_TIM:8 % PMSC_LEDC
    >>);
% mapping pmsc ctrl0 from: https://forum.qorvo.com/t/pmsc-ctrl0-bits8-15/746/3
reg(decode, pmsc, Resp) ->
    % User manual says: reserved bits should be preserved at their reset value =>
        can hardcode their values ? Safe to do that ?
    <<
        Res31:12, BLNKNOW:4, Res15:7, BLNKEN:1, BLINK_TIM:8, % PMSC_LEDC
        TXFINESEQ:16, % PMSC_TXFINESEQ
        _:(25*8), % Reserved 2
        SNOZ_TIM:8, % PMSC_SNOZT
        _:32, % Reserved 1
        KHZCLKDIV:6, _:8, LDERUNE:1, _:1, PLLSYN:1, SNOZR:1, SNOZE:1, ARXSLP:1,
            ATXSLP:1, PKTSEQ:8, _:1, ARX2INIT:1, _:1, % PMSC_CTRL1
        SOFTRESET:4, _:3, PLL2_SEQ_EN:1, KHZCLKEN:1, _:3, GPDRN:1, GPDCE:1, GPRN
            :1, GPCE:1, AMCE:1, _:4, ADCCE:1, OTP:1, Res8:1, Res7:1, FACE:1,
            TXCLKS:2, RXCLKS:2, SYSCLKS:2 % PMSC_CTRL0
    >> = reverse(Resp),
    #{
        pmsc_ledc => #{res31 => Res31, blnknow => BLNKNOW, res15 => Res15, blnken
            => BLNKEN, blink_tim => BLINK_TIM},
        pmsc_txfseq => #{txfineseq => TXFINESEQ},
        pmsc_snozt => #{snoz_tim => SNOZ_TIM},
        pmsc_ctrl1 => #{khzclkdiv => KHZCLKDIV, lderune => LDERUNE, pllsyn =>
```

```
            PLLSYN, snozr => SNOZR, snoze => SNOZE, arxslp => ARXSLP, atxslp =>
            ATXSLP, pktseq => PKTSEQ, arx2init => ARX2INIT},
        pmsc_ctrl0 => #{softreset => SOFTRESET, pll2_seq_en => PLL2_SEQ_EN,
            khzclken => KHZCLKEN, gpdrn => GPDRN, gpdce => GPDCE, gprn => GPRN,
            gpce => GPCE, amce => AMCE, adcce => ADCCE, otp => OTP, res8 => Res8,
            res7 => Res7, face => FACE, txclks => TXCLKS, rxclks => RXCLKS,
            sysclks => SYSCLKS}
    };
reg(decode, RegFile, Resp) -> error({unknown_regfile_to_decode, RegFile, Resp});
reg(encode, RegFile, Resp) -> error({unknown_regfile_to_encode, RegFile, Resp}).

rw(read) -> 0;
rw(write) -> 1.

% Mapping of the different register IDs to their hexadecimal value
regFile(dev_id) -> 16#00;
regFile(eui) -> 16#01;
% 0x02 is reserved
regFile(panadr) -> 16#03;
regFile(sys_cfg) -> 16#04;
% 0x05 is reserved
regFile(sys_time) -> 16#06;
% 0x07 is reserved
regFile(tx_fctrl) -> 16#08;
regFile(tx_buffer) -> 16#09;
regFile(dx_time) -> 16#0A;
% 0x0B is reserved
regFile(rx_fwto) -> 16#0C;
regFile(sys_ctrl) -> 16#0D;
regFile(sys_mask) -> 16#0E;
regFile(sys_status) -> 16#0F;
regFile(rx_finfo) -> 16#10;
regFile(rx_buffer) -> 16#11;
regFile(rx_fqual) -> 16#12;
regFile(rx_ttcki) -> 16#13;
regFile(rx_ttcko) -> 16#14;
regFile(rx_time) -> 16#15;
% 0x16 is reserved
regFile(tx_time) -> 16#17;
regFile(tx_antd) -> 16#18;
regFile(sys_state) -> 16#19;
regFile(ack_resp_t) -> 16#1A;
% 0x1B is reserved
% 0x1C is reserved
regFile(rx_sniff) -> 16#1D;
regFile(tx_power) -> 16#1E;
regFile(chan_ctrl) -> 16#1F;
% 0x20 is reserved
regFile(usr_sfd) -> 16#21;
% 0x22 is reserved
regFile(agc_ctrl) -> 16#23;
regFile(ext_sync) -> 16#24;
regFile(acc_mem) -> 16#25;
regFile(gpio_ctrl) -> 16#26;
regFile(drx_conf) -> 16#27;
regFile(rf_conf) -> 16#28;
% 0x29 is reserved
regFile(tx_cal) -> 16#2A;
regFile(fs_ctrl) -> 16#2B;
regFile(aon) -> 16#2C;
regFile(otp_if) -> 16#2D;
regFile(lde_ctrl) -> regFile(lde_if); % No size ?
```

```
regFile(lde_if) -> 16#2E;
regFile(dig_diag) -> 16#2F;
% 0x30 - 0x35 are reserved
regFile(pmsc) -> 16#36;
% 0x37 - 0x3F are reserved
regFile(RegId) -> error({wrong_register_ID, RegId}).

% Only the writtable subregisters in SRW register files are present here
% AGC_CTRL
subReg(agc_ctrl1) -> 16#02;
subReg(agc_tune1) -> 16#04;
subReg(agc_tune2) -> 16#0C;
subReg(agc_tune3) -> 16#12;
subReg(agc_stat1) -> 16#1E;
subReg(ec_ctrl) -> 16#00;
subReg(gpio_mode) -> 16#00;
subReg(gpio_dir) -> 16#08;
subReg(gpio_dout) -> 16#0C;
subReg(gpio_irqe) -> 16#10;
subReg(gpio_isen) -> 16#14;
subReg(gpio_imode) -> 16#18;
subReg(gpio_ibes) -> 16#1C;
subReg(gpio_iclr) -> 16#20;
subReg(gpio_idbe) -> 16#24;
subReg(gpio_raw) -> 16#28;
subReg(drx_tune0b) -> 16#02;
subReg(drx_tune1a) -> 16#04;
subReg(drx_tune1b) -> 16#06;
subReg(drx_tune2) -> 16#08;
subReg(drx_sfdtoc) -> 16#20;
subReg(drx_pretoc) -> 16#24;
subReg(drx_tune4h) -> 16#26;
subReg(rf_conf) -> 16#00;
subReg(rf_rxctrlh) -> 16#0B;
subReg(rf_txctrl) -> 16#0C;
subReg(ldotune) -> 16#30;
subReg(tc_sarc) -> 16#00;
subReg(tc_pg_ctrl) -> 16#08;
subReg(tc_pgdelay) -> 16#0B;
subReg(tc_pgtest) -> 16#0C;
subReg(fs_pllcfg) -> 16#07;
subReg(fs_plltune) -> 16#0B;
subReg(fs_xtalt) -> 16#0E;
subReg(aon_wcfg) -> 16#00;
subReg(aon_ctrl) -> 16#02;
subReg(aon_rdat) -> 16#03;
subReg(aon_addr) -> 16#04;
subReg(aon_cfg0) -> 16#06;
subReg(aon_cfg1) -> 16#0A;
subReg(otp_wdat) -> 16#00;
subReg(otp_addr) -> 16#04;
subReg(otp_ctrl) -> 16#06;
subReg(otp_stat) -> 16#08;
subReg(otp_rdat) -> 16#0A;
subReg(otp_srdat) -> 16#0E;
subReg(otp_sf) -> 16#12;
subReg(lde_thresh) -> 16#00;
subReg(lde_cfg1) -> 16#806;
subReg(lde_ppindx) -> 16#1000;
subReg(lde_ppampl) -> 16#1002;
subReg(lde_rxantd) -> 16#1804;
subReg(lde_cfg2) -> 16#1806;
```

```
subReg(lde_repc) -> 16#2804;
subReg(evc_ctrl) -> 16#00;
subReg(diag_tmc) -> 16#24;
subReg(pmsc_ctrl0) -> 16#00;
subReg(pmsc_ctrl1) -> 16#04;
subReg(pmsc_snozt) -> 16#0C;
subReg(pmsc_txfseq) -> 16#26;
subReg(pmsc_ledc) -> 16#28.


% Mapping of the size in bytes of the different register IDs
regSize(dev_id) -> 4;
regSize(eui) -> 8;
regSize(panadr) -> 4;
regSize(sys_cfg) -> 4;
regSize(sys_time) -> 5;
regSize(tx_fctrl) -> 5;
regSize(tx_buffer) -> 1024;
regSize(dx_time) -> 5;
regSize(rx_fwto) -> 2; % user manual gives 2 bytes and bits 16-31 are reserved
regSize(sys_ctrl) -> 4;
regSize(sys_mask) -> 4;
regSize(sys_status) -> 5;
regSize(rx_finfo) -> 4;
regSize(rx_buffer) -> 1024;
regSize(rx_fqual) -> 8;
regSize(rx_ttcki) -> 4;
regSize(rx_ttcko) -> 5;
regSize(rx_time) -> 14;
regSize(tx_time) -> 10;
regSize(tx_antd) -> 2;
regSize(sys_state) -> 4;
regSize(ack_resp_t) -> 4;
regSize(rx_sniff) -> 4;
regSize(tx_power) -> 4;
regSize(chan_ctrl) -> 4;
regSize(usr_sfd) -> 41;
regSize(agc_ctrl) -> 33;
regSize(ext_sync) -> 12;
regSize(acc_mem) -> 4064;
regSize(gpio_ctrl) -> 44;
regSize(drx_conf) -> 44; % user manual gives 44 bytes but sum of register length gives 45 bytes
regSize(rf_conf) -> 58; % user manual gives 58 but sum of all its register gives 53 => Placeholder for the remaining 8 bytes
regSize(tx_cal) -> 13; % user manual gives 52 bytes but sum of all sub regs gives 13 bytes
regSize(fs_ctrl) -> 21;
regSize(aon) -> 12;
regSize(otp_if) -> 19; % user manual gives 18 bytes in regs table but sum of all sub regs is 19 bytes
regSize(lde_ctrl) -> undefined; % No size ?
regSize(lde_if) -> undefined; % No size ?
regSize(dig_diag) -> 38; % user manual gives 41 bytes but sum of all sub regs gives 38 bytes
regSize(pmsc) -> 44. % user manual gives 48 bytes but sum of all sub regs gives 41 bytes

%% Gives the size in bytes
subRegSize(agc_ctrl1) -> 2;
subRegSize(agc_tune1) -> 2;
subRegSize(agc_tune2) -> 4;
```

```
subRegSize(agc_tune3) -> 2;
subRegSize(agc_stat1) -> 3;
subRegSize(ec_ctrl) -> 4;
subRegSize(gpio_mode) -> 4;
subRegSize(gpio_dir) -> 4;
subRegSize(gpio_dout) -> 4;
subRegSize(gpio_irqe) -> 4;
subRegSize(gpio_isen) -> 4;
subRegSize(gpio_imode) -> 4;
subRegSize(gpio_ibes) -> 4;
subRegSize(gpio_iclr) -> 4;
subRegSize(gpio_idbe) -> 4;
subRegSize(gpio_raw) -> 4;
subRegSize(drx_tune0b) -> 2;
subRegSize(drx_tune1a) -> 2;
subRegSize(drx_tune1b) -> 2;
subRegSize(drx_tune2) -> 4;
subRegSize(drx_sfdtoc) -> 2;
subRegSize(drx_pretoc) -> 2;
subRegSize(drx_tune4h) -> 2;
subRegSize(rf_conf) -> 4;
subRegSize(rf_rxctrlh) -> 1;
subRegSize(rf_txctrl) -> 4; % ! table in user manual gives 3 but details gives 4
subRegSize(ldotune) -> 5;
subRegSize(tc_sarc) -> 2;
subRegSize(tc_pg_ctrl) -> 1;
subRegSize(tc_pgdelay) -> 1;
subRegSize(tc_pgtest) -> 1;
subRegSize(fs_pllcfg) -> 4;
subRegSize(fs_plltune) -> 1;
subRegSize(fs_xtalt) -> 1;
subRegSize(aon_wcfg) -> 2;
subRegSize(aon_ctrl) -> 1;
subRegSize(aon_rdat) -> 1;
subRegSize(aon_addr) -> 1;
subRegSize(aon_cfg0) -> 4;
subRegSize(aon_cfg1) -> 2;
subRegSize(otp_wdat) -> 4;
subRegSize(otp_addr) -> 2;
subRegSize(otp_ctrl) -> 2;
subRegSize(otp_stat) -> 2;
subRegSize(otp_rdat) -> 4;
subRegSize(otp_srdat) -> 4;
subRegSize(otp_sf) -> 1;
subRegSize(lde_thresh) -> 2;
subRegSize(lde_cfg1) -> 1;
subRegSize(lde_ppindx) -> 2;
subRegSize(lde_ppampl) -> 2;
subRegSize(lde_rxantd) -> 2;
subRegSize(lde_cfg2) -> 2;
subRegSize(lde_repc) -> 2;
subRegSize(evc_ctrl) -> 4;
subRegSize(diag_tmc) -> 2;
subRegSize(pmsc_ctrl0) -> 4;
subRegSize(pmsc_ctrl1) -> 4;
subRegSize(pmsc_snozt) -> 1;
subRegSize(pmsc_txfseq) -> 2;
subRegSize(pmsc_ledc) -> 4;
subRegSize(_) -> error({error}).

%--- Debug -----------------------------------------------------------------

```

```
debug_read(Reg, Value) ->
    io:format("[PmodUWB] read [16#~2.16.0B - ~w] --> ~s -> ~s~n",
        [regFile(Reg), Reg, debug_bitstring(Value), debug_bitstring_hex(Value)]
    ).

debug_write(Reg, Value) ->
    io:format("[PmodUWB] write [16#~2.16.0B - ~w] --> ~s -> ~s~n",
        [regFile(Reg), Reg, debug_bitstring(Value), debug_bitstring_hex(Value)]
    ).
debug_write(Reg, SubReg, Value) ->
    io:format("[PmodUWB] write [16#~2.16.0B - ~w - 16#~2.16.0B - ~w] --> ~s -> ~s~n",
        [regFile(Reg), Reg, subReg(SubReg), SubReg, debug_bitstring(Value), debug_bitstring_hex(Value)]
    ).

debug_bitstring(Bitstring) ->
    lists:flatten([io_lib:format("2#~8.2.0B ", [X]) || <<X>> <= Bitstring]).

debug_bitstring_hex(Bitstring) ->
    lists:flatten([io_lib:format("16#~2.16.0B ", [X]) || <<X>> <= Bitstring]).
```

```
%% @doc This generic module defines the behaviour for any module implementing the transmission of the IEEE 802.15.4 stack
%% @end

-module(gen_mac_tx).

-export([start/2]).
-export([transmit/4]).
-export([stop/2]).

-include("pmod_uwb.hrl").
-include("ieee802154.hrl").
-include("ieee802154_pib.hrl").

%--- Callbacks -----------------------------------------------------------

-callback init(PhyMod::module()) -> State::term().
-callback tx(State::term(),
             Frame::bitstring(),
             Pib  :: pib_state(),
             TxOptions::#tx_opts{}) -> {ok, Newstate::term()}
                                      | {error,
                                         Newstate::term(),
                                         Error::tx_error()}.
-callback terminate(State::term(), Reason::atom()) -> ok.

%--- Types ---------------------------------------------------------------

-export_type([state/0]).

-opaque state() :: {Module::module(), Sub::term()}.

%--- API -----------------------------------------------------------------
-spec start(Module, PhyMod) -> State when
      PhyMod :: module(),
      Module :: module(),
      State  :: state().
start(Module, PhyMod) ->
    {Module, Module:init(PhyMod)}.
```

```

-spec transmit(State, Frame, Pib, TxOptions) -> Result when
      State     :: state(),
      Frame     :: bitstring(),
      Pib       :: pib_state(),
      TxOptions :: tx_opts(),
      Result    :: {ok, State} | {error, State, Error},
      Error     :: tx_error().
transmit({Module, Sub}, Frame, Pib, TxOptions) ->
    case Module:tx(Sub, Frame, Pib, TxOptions) of
        {ok, Sub2} ->
            {ok, {Module, Sub2}};
        {error, Sub2, Error} -> {error, {Module, Sub2}, Error}
    end.

-spec stop(State, Reason) -> ok when
      State  :: state(),
      Reason :: atom().
stop({Module, Sub}, Reason) ->
    Module:terminate(Sub, Reason).
```

```
% @doc This module defines a generic behaviour for duty cycling on the IEEE 802.15.4
%
% The module implementing the behaviour will be responsible to manage the duty cylcing of the IEEE 802.15.4 stack (not the power optimization of the pmod)
% For example, the module implementing this behaviour for a beacon enabled network will have the task to manage the CFP, the CAP and the beacon reception
% When an application will request a transmission the module has to suspend the rx before transmitting
% At the transmission of a frame, the module will have the task to check if there is enough time to transmit the frame (e.g. before the next beacon)
% At the transmisson of a data frame with AR=1 the module has to manage the retransmission of the frame if the ACK isn't correctly received
%  This is because this module will be responsible to check if the retransmission can be done (no beacons or not a CAP) and the reception can't be resumed between retransmission (both are responsabilities of this module)
%
% Beacon enabled
% No transmission during beacon
% TX during CAP
% No TX during CFP unless a slot is attributed to the node
%
% Manage the RX loop (suspend/resume)
%
% @end
-module(gen_duty_cycle).

-include("ieee802154.hrl").
-include("ieee802154_pib.hrl").
-include("pmod_uwb.hrl").

-callback init(PhyModule) -> State when
      PhyModule :: module(),
      State     :: term().
-callback on(State) -> Result when
      State       :: term(),
      Result      :: {ok, State}
                   | {error, State, Error},
      Error       :: atom().
-callback off(State) -> {ok, State} when
```

```
      State :: term().
% Add suspend and resume later
-callback tx(State, Frame, Pib, Ranging) -> Result when
      State       :: term(),
      Frame       :: bitstring(),
      Pib         :: pib_state(),
      Ranging     :: ranging_tx(),
      Result      :: {ok, State, RangingInfo}
                   | {error, State, Error},
      RangingInfo :: ranging_informations(),
      Error       :: tx_error().
-callback terminate(State, Reason) -> ok when
      State  :: term(),
      Reason :: term().

-export([start/2]).
-export([turn_on/1]).
-export([turn_off/1]).
-export([tx_request/4]).
-export([stop/2]).

%--- Types ---------------------------------------------------------------------

-export_type([state/0, input_callback_raw_frame/0]).

-opaque state() :: {Module::module(), Sub::term()}.

-type input_callback_raw_frame() :: fun((Frame                  :: binary(),
                                         LQI                    :: integer(),
                                         UWBPRF                 :: uwb_PRF(),
                                         Security               :: ieee802154:
                                             security(),
                                         UWBPreambleRepetitions ::
                                             uwb_preamble_symbol_repetition(),
                                         DataRate               :: data_rate(),
                                         Ranging                :: ieee802154:
                                             ranging_informations())
                                        -> ok).

%--- API -----------------------------------------------------------------------

% @doc initialize the duty cycle module
% @end
-spec start(Module, PhyModule) -> State when
      Module :: module(),
      PhyModule :: module(),
      State :: state().
start(Module, PhyModule) ->
    {Module, Module:init(PhyModule)}.

% @doc turns on the continuous reception
% @TODO specify which RX module has to be used
-spec turn_on(State) -> Result when
      State    :: state(),
      Result   :: {ok, State} | {error, State, Error},
      Error    :: atom().
turn_on({Mod, Sub}) ->
    case Mod:on(Sub) of
        {ok, Sub2} -> {ok, {Mod, Sub2}};
        {error, Sub2, Error} -> {error, {Mod, Sub2}, Error}
    end.

```

```
% @doc turns off the continuous reception
-spec turn_off(State) -> State when
      State :: state().
turn_off({Mod, Sub}) ->
    {ok, Sub2} = Mod:off(Sub),
    {Mod, Sub2}.

% @doc request a transmission to the duty cycle
% The frame is an encoded MAC frame ready to be transmitted
% If the frame request an ACK, the retransmission is managed by the module
%
% Errors:
% <li> ‘no_ack’: No acknowledgment received after macMaxFrameRetries</li>
% <li> ‘frame_too_long’: The frame was too long for the CAP or GTS</li>
% <li> ‘channel_access_failure’: the CSMA-CA algorithm failed</li>
% @end
-spec tx_request(State, Frame, Pib, Ranging) -> Result when
      State       :: state(),
      Frame       :: bitstring(),
      Pib         :: pib_state(),
      Ranging     :: ranging_tx(),
      State       :: state(),
      Result      :: {ok, State, RangingInfo}
                     | {error, State, Error},
      RangingInfo :: ranging_informations(),
      Error       :: tx_error().
tx_request({Mod, Sub}, Frame, Pib, Ranging) ->
    case Mod:tx(Sub, Frame, Pib, Ranging) of
        {ok, Sub2, RangingInfo} ->
            {ok, {Mod, Sub2}, RangingInfo};
        {error, Sub2, Err} ->
            {error, {Mod, Sub2}, Err}
    end.

% @doc stop the duty cycle module
-spec stop(State, Reason) -> ok when
      State  :: state(),
      Reason :: atom().
stop({Mod, Sub}, Reason) ->
    Mod:terminate(Sub, Reason).
```

```
-module(duty_cycle_non_beacon).

-behaviour(gen_duty_cycle).

% gen_duty_cycle callbacks

-export([init/1]).
-export([on/1]).
-export([off/1]).
-export([tx/4]).
-export([terminate/2]).

% Include

-include("mac_frame.hrl").
-include("ieee802154_pib.hrl").
-include("ieee802154.hrl").

%% @doc
%% The module implementing this behaviour manages the duty cycling of the stack.
```

```
%% This includes:
%% <li>
%%  IEEE 802.15.4 duty cycling:
%%  Beacon enabled network, non-beacon enabled network
%% </li>
%% <li> pmod uwb duty cycling: low power listening, sniff mode </li>
%% @end

%--- Types ---------------------------------------------------------------------


%--- Records -------------------------------------------------------------------
-export_type([state/0]).

-record(state,
        {sniff_ont,
         sniff_offt,
         phy_layer,
         loop_pid,
         mac_tx_state}).
-opaque state() :: #state{}.

%--- gen_duty_cycle callbacks --------------------------------------------------
-spec init(PhyMod) -> State when
      PhyMod :: module(),
      State  :: state().
init(PhyMod) ->
    MacTXState = gen_mac_tx:start(unslotted_CSMA, PhyMod),
    #state{sniff_ont = 3,
           sniff_offt = 4,
           phy_layer = PhyMod,
           loop_pid = undefined,
           mac_tx_state = MacTXState}.

-spec on(State) -> Result when
      State    :: state(),
      Result   :: {ok, State} | {error, State, rx_already_on}.
on(#state{loop_pid = undefined} = State) ->
    LoopPid = rx_loop_on(State),
    {ok, State#state{loop_pid = LoopPid}};
on(State) ->
    {error, State, rx_already_on}.

-spec off(State) -> {ok, State} when
      State :: state().
off(#state{phy_layer = PhyMod, loop_pid = LoopPid} = State) ->
    turn_off_rx_loop(PhyMod, LoopPid, shutdown),
    {ok, State#state{loop_pid = undefined}}.

-spec tx(State, Frame, CsmaParams, Ranging) -> Result when
    State        :: state(),
    Frame        :: binary(),
    CsmaParams   :: pib_state(),
    Ranging      :: ranging_tx(),
    Result       :: {ok, State, RangingInfo}
                  | {error, State, Error},
    RangingInfo :: ranging_informations(),
    Error        :: tx_error().
tx(#state{loop_pid = undefined} = State, Frame, CsmaParams, Ranging) ->
    case tx_(State, Frame, CsmaParams, Ranging) of
        {ok, NewMacTxState} ->
            RangingInfo = tx_ranging_infos(Ranging, State),
```

```erlang
                {ok, State#state{mac_tx_state = NewMacTxState}, RangingInfo};
            {error, NewMacTxState, Error} ->
                {error, State#state{mac_tx_state = NewMacTxState}, Error}
        end;
tx(State, Frame, CsmaParams, Ranging) ->
    suspend_rx_loop(State),
    TxStatus = tx_(State, Frame, CsmaParams, Ranging),
    NewLoopPid = rx_loop_on(State),
    case TxStatus of
        {ok, NewMacTxState} ->
            RangingInfo = tx_ranging_infos(Ranging, State),
            {ok,
             State#state{loop_pid = NewLoopPid,
                         mac_tx_state = NewMacTxState},
            RangingInfo};
        {error, NewMacTxState, Error} ->
            {error, State#state{loop_pid = NewLoopPid,
                                mac_tx_state = NewMacTxState},
             Error}
    end.

-spec terminate(State, Reason) -> ok when
      State :: state(),
      Reason :: atom().
terminate(State, Reason) ->
    LoopPid = State#state.loop_pid,
    PhyMod = State#state.phy_layer,
    MacTXState = State#state.mac_tx_state,
    turn_off_rx_loop(PhyMod, LoopPid, Reason),
    gen_mac_tx:stop(MacTXState, Reason),
    ok.

%--- internal -------------------------------------------------------------

%% @doc loop function for the reception
%% This function waits for a reception event to occur
%% If the event is the reception of a frame,
%% it will call the callback function to notify the next higher level/layer
%% If the event is an error, the function ignores it
%% @end
-spec rx_loop(PhyMod) -> no_return() when
      PhyMod :: module().
rx_loop(PhyMod) ->
    PhyMod:reception_async(),
    rx_loop(PhyMod).

%% @doc
%% Sets the settings for reception and turns on the reception loop process
%% Returns the pid of the loop process
%% Note: this function will change when OS interrupts are introduced
%% @end
-spec rx_loop_on(State) -> pid() when
      State      :: state().
rx_loop_on(State) ->
    PhyMod = State#state.phy_layer,
    SniffOnT = State#state.sniff_ont,
    SniffOffT = State#state.sniff_offt,
    PhyMod:write(tx_fctrl, #{tr => 1}),
    PhyMod:write(sys_cfg, #{rxwtoe => 0}),
    PhyMod:write(pmsc, #{pmsc_ctrl1 => #{arx2init => 2#1}}),
    PhyMod:write(rx_sniff, #{sniff_ont => SniffOnT, sniff_offt => SniffOffT}),
    spawn_link(fun() -> rx_loop(PhyMod) end).
```

```erlang

-spec turn_off_rx_loop(PhyMod, LoopPid, Reason) -> ok when
      PhyMod  :: module(),
      LoopPid :: pid() | undefined,
      Reason  :: atom().
turn_off_rx_loop(_, undefined, _) -> ok;
turn_off_rx_loop(PhyMod, LoopPid, Reason) ->
    PhyMod:disable_rx(),
    unlink(LoopPid),
    exit(LoopPid, Reason),
    PhyMod:write(pmsc, #{pmsc_ctrl1 => #{arx2init => 2#0}}),
    PhyMod:write(rx_sniff, #{sniff_ont => 2#0, sniff_offt => 2#0}),
    PhyMod:disable_rx(),
    PhyMod:write(sys_cfg, #{rxwtoe => 1}).

-spec suspend_rx_loop(State) -> ok when
      State :: state().
suspend_rx_loop(State) ->
    PhyMod = State#state.phy_layer,
    LoopPid = State#state.loop_pid,
    turn_off_rx_loop(PhyMod, LoopPid, shutdown).

% @private
-spec tx_(State, Frame, Pib, Ranging) -> Result when
      State   :: state(),
      Frame   :: bitstring(),
      Pib     :: pib_state(),
      Ranging :: ranging_tx(),
      Result  :: {ok, MacTXState} | {error, MacTXState, Error},
      Error   :: atom().
tx_(State, <<_:2, ?ENABLED:1, _:13, Seqnum:8, _/binary>> = Frame, Pib, Ranging) ->
    MacTXState = State#state.mac_tx_state,
    PhyMod = State#state.phy_layer,
    tx_ar(MacTXState, PhyMod, Frame, Seqnum, 0, Pib, Ranging);
tx_(State, Frame, CsmaParams, Ranging) ->
    MacTXState = State#state.mac_tx_state,
    TxOpts = #tx_opts{ranging = Ranging},
    gen_mac_tx:transmit(MacTXState, Frame, CsmaParams, TxOpts).

% @private
% @doc This function transmits a frame with AR=1
% If the ACK isn't received before the timeout, a retransmission is done
% If the frame has been transmitted MACMAXFRAMERETRIES times then the error
% 'no_ack' is returned
% @end
-spec tx_ar(MacTXState, PhyMod, Frame, Seqnum, Retry, Pib, Ranging) -> Result when
      MacTXState :: gen_mac_tx:state(),
      PhyMod     :: module(),
      Frame      :: bitstring(),
      Seqnum     :: non_neg_integer(),
      Retry      :: non_neg_integer(),
      Pib        :: pib_state(),
      Ranging    :: ranging_tx(),
      Result     :: {ok, MacTxState} | {error, MacTxState, Error},
      Error      :: atom().
tx_ar(MacTxState, _, _, _, ?MACMAXFRAMERETRIES, _, _) ->
    {error, MacTxState, no_ack};
tx_ar(MacTXState, PhyMod, Frame, Seqnum, Retry, Pib, Ranging) ->
    TxOpts = #tx_opts{wait4resp = ?ENABLED, ranging = Ranging},
    case gen_mac_tx:transmit(MacTXState, Frame, Pib, TxOpts) of
        {ok, NewMacTxState} ->
            case wait_for_ack(PhyMod, Seqnum) of
```

```
                ok ->
                    {ok, NewMacTxState};
                no_ack ->
                    tx_ar(NewMacTxState,
                          PhyMod,
                          Frame,
                          Seqnum,
                          Retry+1,
                          Pib,
                          Ranging)
            end;
        {error, NewMacTxState, _Error} ->
            tx_ar(NewMacTxState,
                  PhyMod,
                  Frame,
                  Seqnum,
                  Retry+1,
                  Pib,
                  Ranging)
    end.

wait_for_ack(PhyMod, Seqnum) ->
    case PhyMod:reception(true) of
        {_, <<_:5, ?FTYPE_ACK:3, _:8/bitstring, Seqnum:8>>} ->
            ok;
        {_, <<_:5, FType:3, _/bitstring>>} = Frame when FType =/= ?FTYPE_ACK ->
            ieee802154_events:rx_event(Frame, PhyMod:get_rx_metadata()),
            no_ack;
        _ ->
            no_ack
    end.

%--- Internal: Ranging helpers

tx_ranging_infos(?NON_RANGING, _) ->
    #ranging_informations{ranging_received = false};
tx_ranging_infos(?ENABLED, State) ->
    #state{phy_layer = PhyMod} = State,
    #{rx_stamp := RxStamp} = PhyMod:read(rx_time),
    #{tx_stamp := TxStamp} = PhyMod:read(tx_time),
    #{rxtofs := RXTOFS} = PhyMod:read(rx_ttcko),
    #{rxttcki := RXTTCKI} = PhyMod:read(rx_ttcki),
    #ranging_informations{
       ranging_received = true,
       ranging_counter_start = TxStamp,
       ranging_counter_stop = RxStamp,
       ranging_tracking_interval = RXTTCKI,
       ranging_offset = RXTOFS,
       ranging_FOM = <<0:8>>
      }.
```

```
-record(device, {slot, driver, pid, monitor}).
```

```
% This module has the responsability of managing the channel access (CSMA/CA
    algorithm)
-module(unslotted_CSMA).

-include("ieee802154.hrl").
-include("ieee802154_pib.hrl").
-include("pmod_uwb.hrl").

```

```
-behaviour(gen_mac_tx).

-export([init/1]).
-export([tx/4]).
-export([terminate/2]).

%--- Macros -----------------------------------------------------------------

% According to Qorov forums, 1 symbol ~ 1  s  => The unit of AUNITBACKOFFPERIOD are in  s
-define(AUNITBACKOFFPERIOD, 20). % The number of symbols forming the basic time period used in CSMA-CA (src. IEEE 802.15.4 stdMA-CA (src. IEEE 802.15.4 std.)


%% CCA Mode 5 should last at least the maximum packet duration + the maximum period for ACK
%% Maximum packet duration = 1207.79 s
%% Maximum period for ACK = 1058.21 s  + 12 s
%% Sum => 2272 s
%% Since PRETOC units are in PAC size, we know that the default PAC is 8 symbols and 1 symbol ~ 1 s
%% We can conclude that CCA_DURATION = ceil(2272/8) = 284
% -define(CCA_DURATION, 284).

%--- Records ----------------------------------------------------------------

%--- gen_mac_tx Callbacks ---------------------------------------------------

-spec init(PhyMod) -> State when
      PhyMod :: module(),
      State  :: map().
init(PhyMod) ->
    #{phy_layer => PhyMod}.

%% @doc Tries to transmit a frame using unslotted CSMA-CA
%% @param MacMinBE: The minimum value of the backoff exponent as described in the standard
%% @param MacMaxCSMABackoffs: The maximum amount of time the CSMA algorithm will backoff if the channel is busy
%% @param CW0: Not needed in this version of the algorithm. Ignored by this function
-spec tx(State, Frame, Pib, TxOpts) -> {ok, State} | {error, State, channel_access_failure} when
      State  :: map(),
      Frame  :: bitstring(),
      Pib    :: pib_state(),
      TxOpts :: tx_opts().
tx(#{phy_layer := PhyMod} = State, Frame, Pib, TxOpts) ->
    CCADuration = math:ceil(cca_duration(PhyMod)),
    PhyMod:write(sys_cfg, #{autoack => 0}),
    MacMinBE = ieee802154_pib:get(Pib, mac_min_BE),
    MacMaxBE = ieee802154_pib:get(Pib, mac_max_BE),
    MacMaxCSMABackoffs = ieee802154_pib:get(Pib, mac_max_csma_backoffs),
    PhyMod:set_frame_timeout(CCADuration),
    Ret = case try_cca(PhyMod, 0, MacMinBE, MacMaxBE, MacMaxCSMABackoffs) of
              ok ->
                  PhyMod:transmit(Frame, TxOpts),
                  {ok, State};
              error ->
                  {error, State, channel_access_failure}
          end,
    PhyMod:write(sys_cfg, #{autoack => 1}),
```

```
    Ret.

terminate(_State, _Reason) -> ok.

%--- Internal ------------------------------------------------------------

% @doc Tries CCA until NB > maxCSMABackoff of if channel is detected idle
%
% The algorithm is described in figure 11 in sec. 5.1.1.4
%
% The timing settings to perform CCA shall be set prior to calling this func.
% @end
-spec try_cca(PhyMod, NB, BE, MacMaxBE, MacMaxCSMABackoffs) -> Result when
      PhyMod              :: module(),
      NB                  :: non_neg_integer(),
      BE                  :: non_neg_integer(),
      MacMaxBE            :: mac_max_BE(),
      MacMaxCSMABackoffs  :: mac_max_csma_backoff(),
      Result              :: ok | error.
try_cca(_, NB, _, _, MacMaxCSMABackoffs) when NB > MacMaxCSMABackoffs ->
    error;
try_cca(PhyMod, NB, BE, MacMaxBE, MacMaxCSMABackoffs) ->
    PhyCfg = PhyMod:get_conf(),
    RandBackOff = ieee802154_utils:symbols_to_usec(random_backoff(BE), PhyCfg),
    SleepTime = trunc(math:ceil(RandBackOff/1000)),
    timer:sleep(SleepTime),
    case cca(PhyMod) of
        ok ->
            ok;
        error ->
            try_cca(PhyMod, NB+1, min(BE+1,MacMaxBE), MacMaxBE, MacMaxCSMABackoffs
                )
    end.

% @doc Performs CCA
-spec cca(PhyMod) -> Result when
      PhyMod :: module(),
      Result :: ok | error.
cca(PhyMod) ->
    case PhyMod:reception() of
        {error, rxrfto} -> ok;
        {error, rxprd} -> error;
        {error, rxsfdd} -> error; % theoritically, this should cover any frame rx
            (i.e. channel is busy)
        _ -> error % In case you receive a frame -> ? Could this happen ?
    end.

% @doc Give the CCA duration in micro-seconds for mode 5
% According to sec. 8.2.7 the CCA period shall be no shorter than
% The maximum packet duration + maximum period for acknowledgment
%
% @end
cca_duration(PhyMod) ->
    Conf = PhyMod:get_conf(),
    TMaxPckt = ieee802154_utils:pckt_duration(127, Conf),
    TAckPckt = ieee802154_utils:pckt_duration(5, Conf),
    TurnAroundRxTx = 12, % us cf. datasheet sec. 5.1.6
     ((TMaxPckt + TAckPckt) / ieee802154_utils:t_dsym(Conf)) + TurnAroundRxTx.

% @doc computes the backoff period (in symbol units)
% Table 11 - sec.5.1.1.4 says that this period shall be equal to:
% $$ \text{random}(2^{BE} - 1) $$ backoff units
```

```
% To get the value is symbol units => multiply the result by AUNITBACKOFFPERIOD
random_backoff(BE) ->
    Backoff = round(math:pow(2, BE)) - 1, % [0, 2^BE-1]
    rand:uniform(Backoff * ?AUNITBACKOFFPERIOD).
    %rand:uniform(max(Backoff * ?AUNITBACKOFFPERIOD, 6000)).
```

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**