

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

1	Introduction	1
1.1	6LoWPAN features	2
1.2	Thesis context	3
1.3	Contributions	3
1.4	Document structure	4
2	State of the art	5
2.1	Localization techniques and UWB	5
2.1.1	WiFi and BLE	5
2.1.2	Zigbee and Acoustic Ultrasound	6
2.1.3	Global Navigation Satellite System (GNSS)	6
2.1.4	Ultra-Wideband (UWB)	6
2.2	Embedded systems programming	6
2.3	Communication protocols and 6LoWPAN	7
3	Hardware description	8
3.1	GRiSP 2 board	8
3.2	Pmod sensor - DW1000	9
3.3	UWB sniffer	9
4	Previous work	11
4.1	Mac layer usage and API	12
5	6LoWPAN	13
5.1	Overview	13
5.2	RFC 4944	14
5.2.1	IEEE 802.15.4 mode for IP	14
5.2.2	Frame encapsulation and formats	14
5.2.3	Dispatch type and header	16
5.2.4	Meshing	16
5.2.5	Fragmentation	17

5.2.6	Stateless Address Auto-configuration	18
5.2.7	IPv6 link local address	18
5.2.8	Multicast address mapping	19
5.3	RFC 6282	19
5.3.1	Header compression	19
5.3.2	LOWPAN_IPHC encoding format	20
5.3.3	Encoding	20
5.3.4	Context Identifier extension	23
5.3.5	UDP header compression	23
6	Design and implementation	24
6.1	Code architecture	24
6.1.1	Code architecture overview	24
6.2	Lowpan API	26
6.2.1	Generic state machine	26
6.2.2	Lowpan API state machine diagram	28
6.3	Lowpan API	31
6.3.1	Packet transmission algorithm	31
6.3.2	Frame reception algorithm	32
6.4	6LoWPAN core	33
6.4.1	Header compression	33
6.4.2	Next header compression	33
6.4.3	Fragmentation	34
6.4.4	Meshing	35
6.4.5	Stateless address generation	36
6.4.6	Reassembly	36
6.4.7	Header decoding	36
6.5	Routing table	37
6.5.1	gen_server	37
7	Tests and results	38
7.1	Hardware constraints	38
7.2	Default field values	38
7.3	Software tests	39
7.3.1	Function validation tests	39
7.3.2	Simulation testing	40
7.4	Hardware tests	45
7.4.1	Robot application	46
7.4.2	Testing environment setup	46
7.4.3	Packet format validation	47
7.4.4	Two GRiSPs communication	52

7.4.5	Three GRiSPs communication	54
7.4.6	Five GRiSPs routing	56
7.5	Contiki-ng validation tests	57
7.5.1	Setup	57
7.5.2	First example	58
7.5.3	Second example	59
7.5.4	Third example	59
7.5.5	Observations	60
8	Future work	61
9	Conclusion	62
A	Appendix	68
A.1	General Note	68
A.2	Methodology roadmap	68
A.3	Lowpan header file code	68
A.4	Lowpan core code	73
A.5	Lowpan API code	101
A.6	Routing table code	116
A.7	Lowpan ipv6 code	117
A.8	Utils file for testing code	119
A.9	Functional testing code	120
A.10	Simulation tests code	135
A.11	Robot application code	154
A.12	Mockups files code	163
A.13	Mac layer code	179

List of Figures

1.1	6LoWPAN stack	2
1.2	IEEE 802.15.4-2011 mac header	2
3.1	GRiSP 2 board [1]	8
3.2	Pmod uwb	9
3.3	UWB Sniffer operation [2]	10
5.1	Encapsulation of 6LoWPAN datagram in a MAC frame	14
5.2	Encapsulated IPv6 datagram format	15
5.3	Compressed IPv6 datagram format	15
5.4	Compressed and meshed IPv6 datagram format	15
5.5	Compressed and fragmented IPv6 datagram	15
5.6	Compressed, fragmented and meshed IPv6 datagram format	15
5.7	Compressed and broadcasted IPv6 datagram format	15
5.8	Mesh addressing header	16
5.9	First fragment header	17
5.10	Subsequent fragment header	17
5.11	Pseudo 48-bit address	18
5.12	LOWPAN_IPHC encoding format	20
5.13	Context Identifier encoding format	23
6.1	Code architecture	25
6.2	Lowpan API state diagram	30
7.1	Functional testing results	40
7.2	Software test workflow	41
7.3	Simple transmission Sender	41
7.4	Simple transmission Receiver	41
7.5	Big packet transmission	42
7.6	Meshed transmission Sender	43
7.7	Meshed transmission Forwarder	43
7.8	Meshed transmission Receiver	43

7.9	Timeout scenario Sender	44
7.10	Timeout scenario Receiver	44
7.11	Duplicate transmission Sender	45
7.12	Duplicate transmission Receiver	45
7.13	Wireshark setup 1	46
7.14	6LoWPAN wirehsark activation	47
7.15	Wireshark setup 2	47
7.16	Hardware test workflow	47
7.17	Uncompressed IPv6 datagram encapsulation, Wireshark pcap	48
7.18	Compressed IPv6 datagram encapsulation, Wireshark pcap	48
7.19	Meshed IPv6 datagram encapsulation, Wireshark pcap	49
7.20	Fragmented IPv6 datagram encapsulation, Wireshark pcap	50
7.21	Compressed, meshed and fragmented encapsulation, Wireshark pcap	50
7.22	Broadcasted IPv6 datagram encapsulation, Wireshark pcap	51
7.23	UDP encapsulation, Wireshark pcap	51
7.24	Simple communication GRiSP Sender	52
7.25	Simple communication GRiSP Receiver	52
7.26	Big payload transmission GRiSP Sender	53
7.27	Big payload transmission GRiSP Receiver	53
7.28	Three nodes routing test Sender	55
7.29	Three nodes routing test Receiver	55
7.30	Routing test Sender	55
7.31	Routing test Forwarder	55
7.32	Routing test Receiver	55
7.33	Five nodes routing test	56
7.34	Five nodes routing Node 1	57
7.35	Five nodes routing Node 3	57
7.36	Five nodes routing Node 5	57
7.37	UDP border router cooja simulation	58
7.38	Contiki 1st example Wireshark capture	58
7.39	1st example compression output	58
7.40	Contiki 2nd example Wireshark capture	59
7.41	2nd example compression output	59
7.42	Contiki 3rd example Wireshark capture	60
7.43	3rd example compression output	60

List of Tables

5.1	Non-Reserved dispatch value bit pattern	16
5.2	Traffic class and Flow label compression	20
5.3	Next Header compression	20
5.4	Hop limit compression	21
5.5	Context Identifier encoding	21
5.6	SAC and DAC encoding	21
5.7	Source Address Mode encoding	22
5.8	Multicast address encoding	22
5.9	Destination Address Mode encoding	22
5.10	UDP Checksum encoding	23
5.11	UDP Ports encoding	23
7.1	Default field values	38
7.2	Function validation tests	39
7.3	Sniffer configuration settings	46
7.4	Station communication tests	54
7.5	1st Contiki example IPv6 field values	58
7.6	2nd Contiki example IPv6 field values	59
7.7	3rd Contiki example IPv6 field values	59

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.

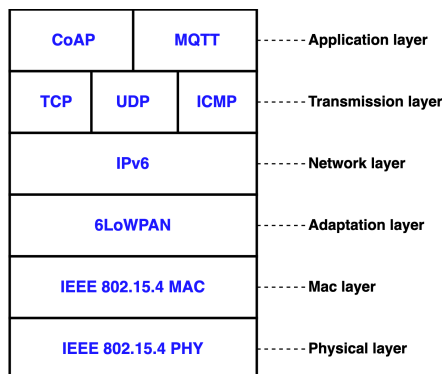


Figure 1.1: 6LoWPAN stack

1.1 6LoWPAN features

To benefit 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]

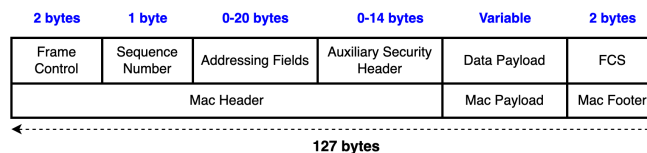


Figure 1.2: IEEE 802.15.4-2011 mac header

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 a 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, developed 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 Machine (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]

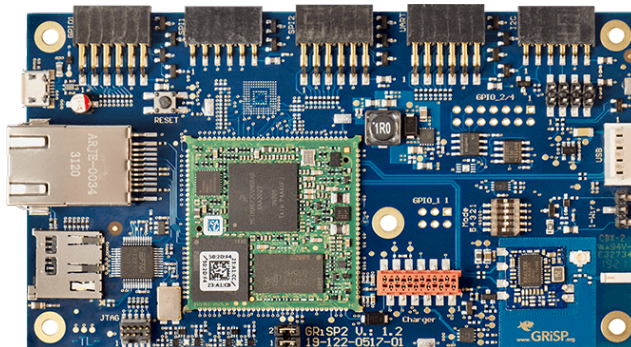


Figure 3.1: GRiSP 2 board [1]

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]

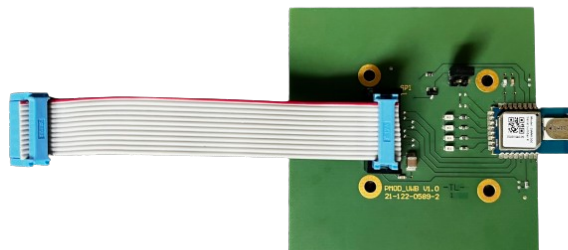


Figure 3.2: Pmod uwb

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]

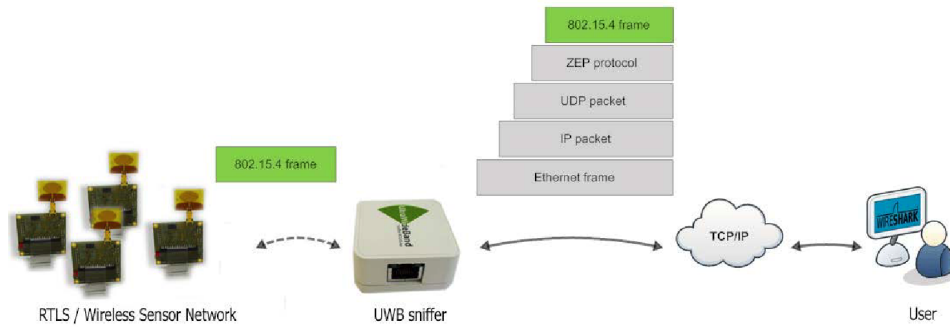


Figure 3.3: UWB Sniffer operation [2]

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.

```
1 -spec set_pib_attribute(Attribute, Value) -> ok when
2   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.

```
1 -spec start(Params) -> {ok, pid()} | {error, any()} when
2   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.

```
1 -spec transmission(Frame) -> {ok, map()} | {error, Error} when
2   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.

```
1 -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.

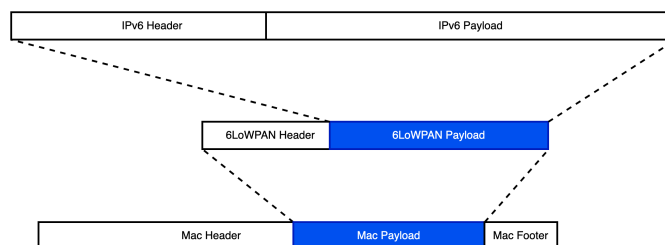


Figure 5.1: Encapsulation of 6LoWPAN datagram in a MAC frame

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:



Figure 5.2: Encapsulated IPv6 datagram format

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

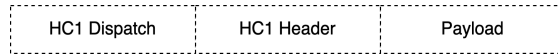


Figure 5.3: Compressed IPv6 datagram format

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

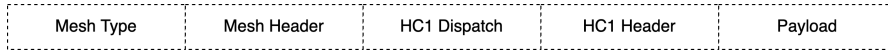


Figure 5.4: Compressed and meshed IPv6 datagram format

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

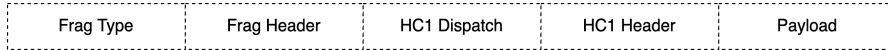


Figure 5.5: Compressed and fragmented IPv6 datagram

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

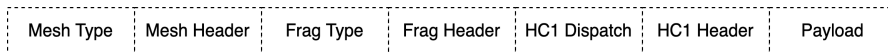


Figure 5.6: Compressed, fragmented and meshed IPv6 datagram format

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

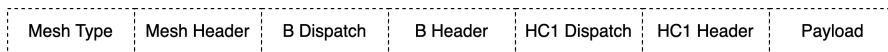


Figure 5.7: Compressed and broadcasted IPv6 datagram format

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.

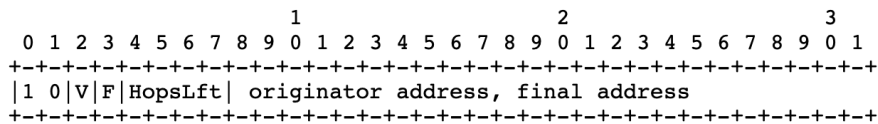


Figure 5.8: Mesh addressing header

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.

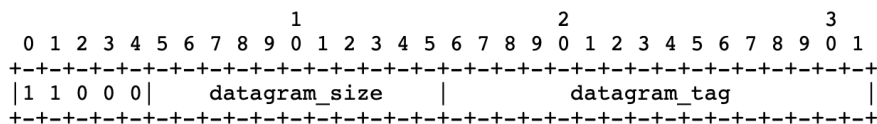


Figure 5.9: First fragment header

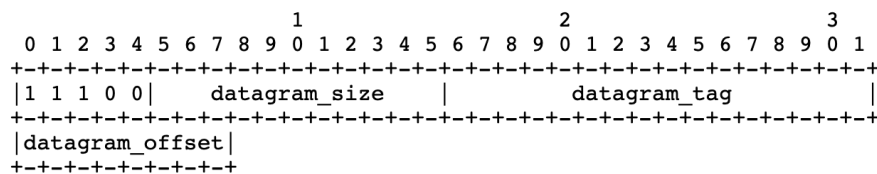


Figure 5.10: Subsequent fragment header

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.

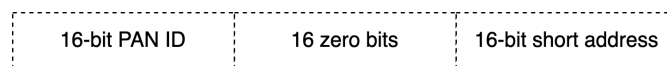


Figure 5.11: Pseudo 48-bit address

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

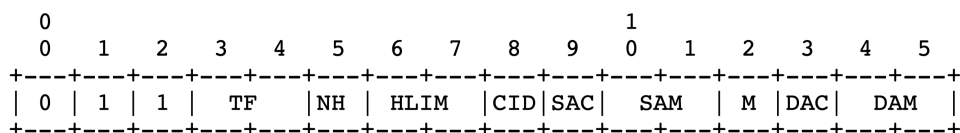


Figure 5.12: LOWPAN_IPHC encoding format

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/Destination 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 <code>fe80::/64</code>
0	10	Last 16 bits in-line; first 112 bits are <code>fe80::0000:00ff:fe00</code>
0	11	Address fully elided; derived from link-local prefix and header
1	00	UNSPECIFIED address, <code>::</code>
1	01	Last 64 bits in-line; prefix derived from context
1	10	Last 16 bits in-line; prefix from context, IID from <code>::ff:fe00:XXXX</code>
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 <code>fe80::/64</code>
0	0	10	Last 16 bits in-line; first 112 bits are <code>fe80::0000:00ff:fe00</code>
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 <code>::ff:fe00:XXXX</code>
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 <code>ffXX::00XX:XXXX:XXXX</code>
1	0	10	Last 32 bits in-line; address in form <code>ffXX::00XX:XXXX</code>
1	0	11	Last 8 bits in-line; address in form <code>ff02::00XX</code>
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.

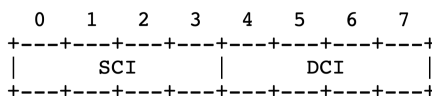


Figure 5.13: Context Identifier encoding format

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.

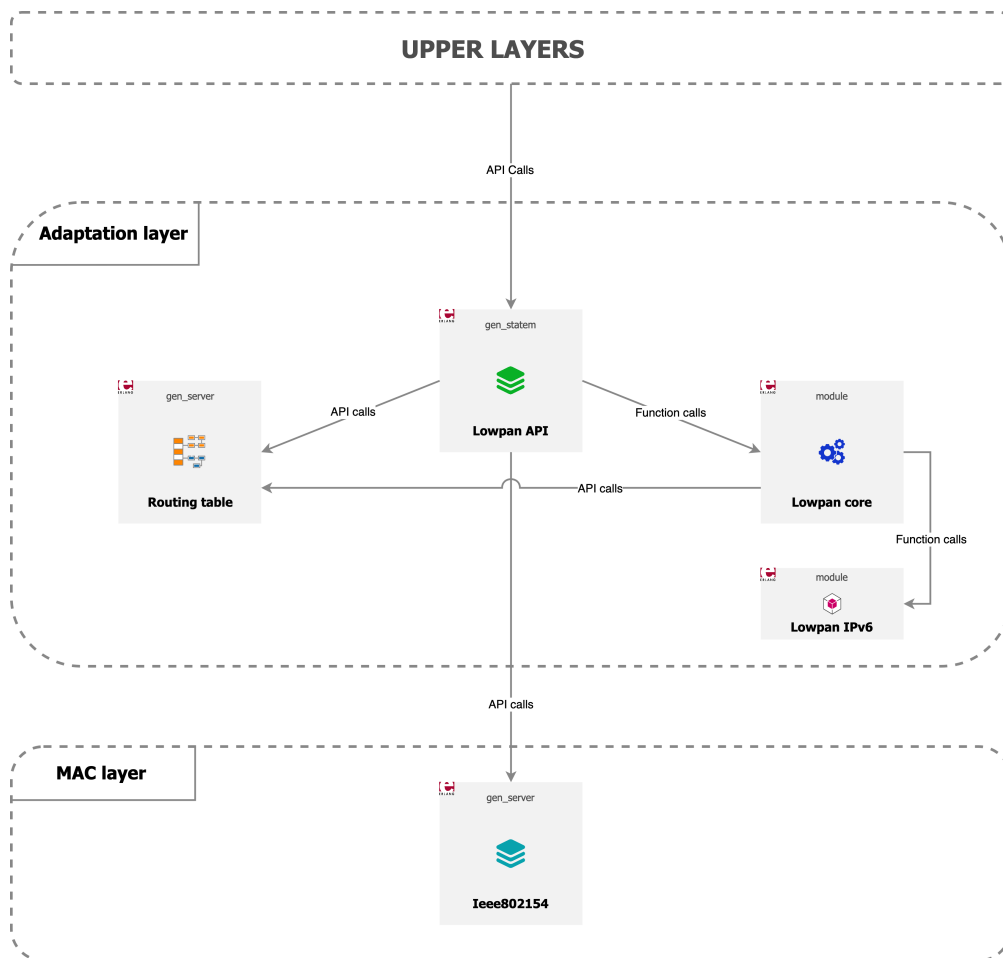


Figure 6.1: Code architecture

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.

```

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

```

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

```

1 handle_event(EventType, EventContent, State, Data) ->
2   % 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.

```

1 idle(cast, {frame_rx, From}, _Data)-> % Handle asynchronous 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.

```

1 {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.

```

1 {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.

```
1 start_link(Params) ->
2   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 .

```
1 stop() ->
2   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.

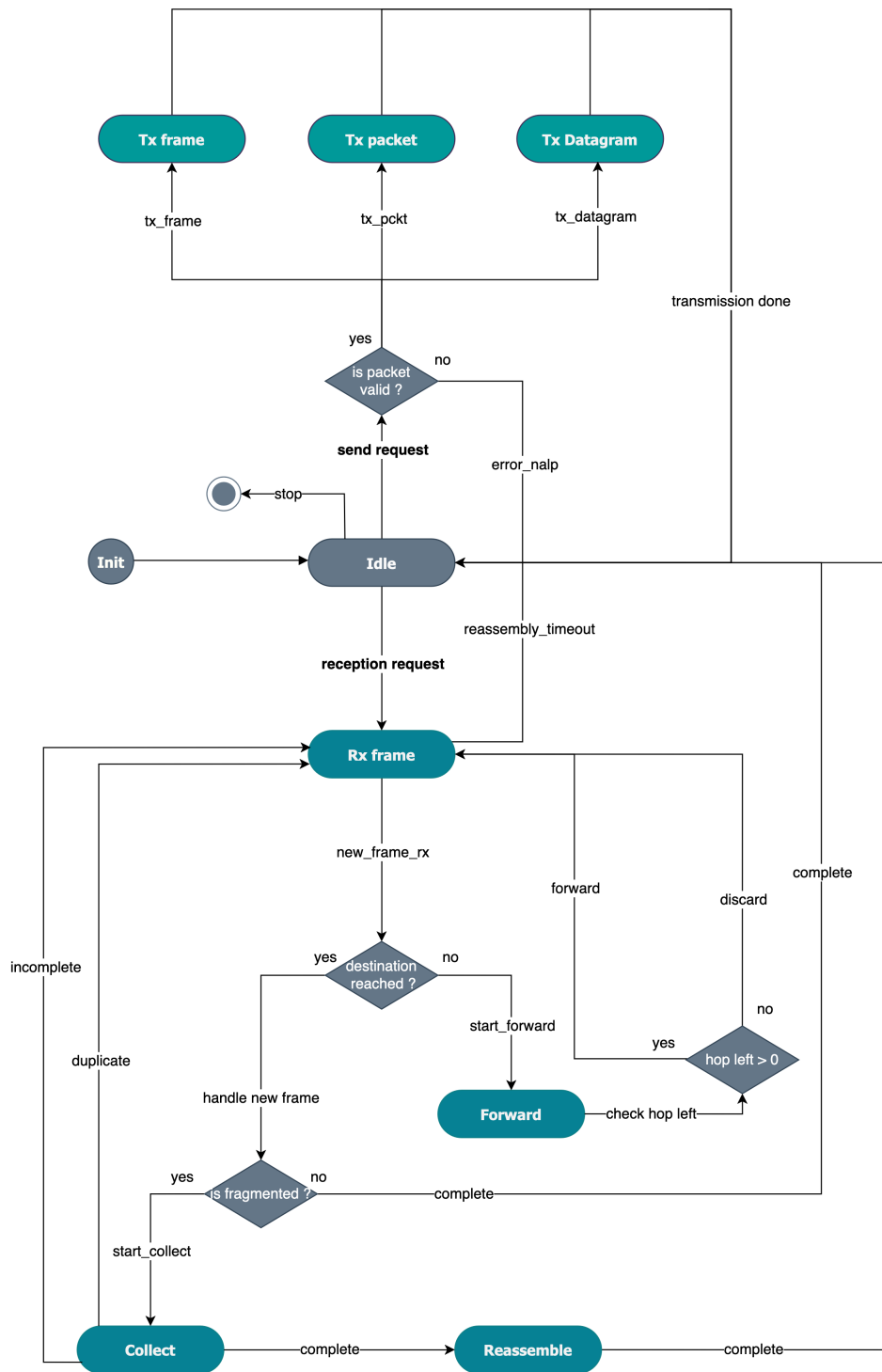


Figure 6.2: Lowpan API state diagram

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 Transmission of an IPv6 packet

Implements: LowpanAPI, **instance** *lowpan_api*

Uses: LowpanCore, **instance** *lowpan_core*; RoutingTable, **instance** *routing_table*;

```
upon event  $\langle lowpan\_api, tx\_pkt \mid IPv6Pckt, PcktInfo, ExtendedHopsLeft, From \rangle$  do
  {DestMacAddr, SenderMacAddr}  $\leftarrow lowpan\_api: getEUI64MacAddr$ 
  MacAddrs  $\leftarrow \{DestMacAddr, SenderMacAddr\}$ 
  {RouteExist, MeshedHdrBin, MH}  $\leftarrow lowpan\_core: getNextHop(PcktInfo, MacAddrs)$ 
  CompressedHeader  $\leftarrow lowpan\_core: compressIPv6Header(IPv6Pckt, RouteExist)$ 
  CompressedPckt  $\leftarrow buildPacket(CompressedHeader, PcktInfo)$ 
  {FragReq, Fragments}  $\leftarrow lowpan\_core: triggerFragmentation(CompressedPckt, RouteExist, Tag)$ 
  if FragReq == TRUE then
    Response  $\leftarrow sendFragments(Fragments, MeshedHdrBin, MH)$ 
    reply Response to From
  else if FragReq == FALSE then
    Response  $\leftarrow sendFragment(Fragments, MeshedHdrBin, MH)$ 
    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 introduced 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, MeshPcktInfo} ← 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.

```
1 encodeDam(_CID, 1, 1, DstAdd, CarrInlineMap, CarrInlineList, _RouteExist) ->
2   DestAddBits = <<DstAdd:128>>,
3   <<_:80, Last48Bits:48>> = DestAddBits,
4   case DestAddBits of
5     <<16#FF, _:112>> ->
6       Bin = <<Last48Bits:48>>,
7       L = [Bin],
8       UpdatedList = [CarrInlineList, L],
9       UpdatedMap = CarrInlineMap#{"DAM" => Bin},
10      {2#00, UpdatedMap, UpdatedList}
11   end.
```

The header compression code is given in the appendix.

6.4.2 Next header compression

The Next header compression is performed using 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.

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

```

2   case {<<SrcPort:16>>, <<DstPort:16>>} of
3     {<<?0xf0b:12, Last4S_Bits:4>>, <<?0xf0b:12, Last4D_Bits:4>>} ->
4       ToCarr = <<Last4S_Bits:4, Last4D_Bits:4>>,
5       L = [ToCarr],
6       CarriedInlineList = CarriedInline ++ L,
7       P = 2#11,
8       {P, CarriedInlineList}
9   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 and 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: `CurrNodeMacAddr`, `SenderMacAddr`, `DestMacAddress`, `DestAddress`, `SeqNum`, `Hopsleft_extended`
Output: `{isMeshed, Header, MHdr}`
if `DestAddressPrefix == Multicast` **then**
 `MulticastAddr` \leftarrow `lowpan_core:generateMulticastAddr(DestAddress)`
 `MulticastEU64` \leftarrow `lowpan_core:generateEUI64MacAddr(MulticastAddr)`
 `MH` \leftarrow `createMacHeader(CurrNodeMacAddr, MulticastEU64)`
 `BroadcastHeader` \leftarrow `lowpan_core:createBroadcastHeader(SeqNum)`
 `MeshHdrBin` \leftarrow `lowpan_core:createNewMeshHeader(SenderMacAddr, DestMacAddress, Hopsleft_extended)`
 `Header` \leftarrow `concat(MeshHdrBin, BroadcastHeader)`
 return `{false, Header, MH}`
else
 `NextHopMacAddr` \leftarrow `routing_table:getRoute(DestMacAddress)`
 if `NextHopMacAddr \neq DestMacAddress` **then**
 `MH` \leftarrow `createMacHeader(CurrNodeMacAddr, NextHopMacAddr)`
 `MeshHdrBin` \leftarrow `lowpan_core:createNewMeshHeader(SenderMacAddr, DestMacAddress, Hopsleft_extended)`
 return `{true, MeshHdrBin, MacHdr}`
 else if `NextHopMacAddr == DestMacAddress` **then**
 `MHdr` \leftarrow `{src_addr = CurrNodeMacAddr, 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.

```

1 -spec generateEUI64MacAddr(binary()) -> binary().
2 generateEUI64MacAddr(MacAddr) when byte_size(MacAddr) == ?SHORT_ADDR_LEN ->
3   PanID = <<16#FFFF:16>>,
4   Extended48Bit = <<PanID/binary, 0:16, MacAddr/binary>>,
5   <<A:8, Rest:40>> = Extended48Bit,
6   ULBSetup = A band 16#FD,
7   <<First:16, Last:24>> = <<Rest:40>>,
8   EUI64 = <<ULBSetup:8, First:16, 16#FF:8, 16#FE:8, Last:24>>,
9   EUI64;
10 generateEUI64MacAddr(MacAddr) when byte_size(MacAddr) == ?EXTENDED_ADDR_LEN
11 ->
12   <<A:8, Rest:56>> = MacAddr,
13   NewA = A bxor 2,
14   <<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.

```

1 decodeTf(TF, CarriedInline) ->
2   case TF of
3     2#10 -> <<ECN:2, DSCP:6, Rest/bitstring>> = CarriedInline,

```

```
4     {{DSCP, ECN}, 0, Rest}
5     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.

```
1 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.

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

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

```
1 stop() ->
2     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
<code>pkt_encapsulation_test</code>	Tests the encapsulation of an IPv6 packet by verifying that it is correctly wrapped with the appropriate 6LoWPAN header.
<code>iphc_pkt_64bit_addr</code>	Tests the IPv6 header compression function when using 64-bit addresses.
<code>msh_pkt</code>	Validates the construction of a 6LoWPAN mesh header.
<code>link_local_addr_pkt_comp</code>	Ensures the compression of IPv6 headers using link-local addresses is performed correctly, reducing the header size as expected.
<code>multicast_addr_pkt_comp</code>	Validates that IPv6 headers containing multicast addresses are compressed properly.
<code>global_context_pkt_comp1</code>	Tests the compression of IPv6 headers using global context address.
<code>udp_nh_pkt_comp</code>	Checks the compression of IPv6 headers where UDP is the next header, ensuring correct compression and inclusion of UDP-specific fields.
<code>compress_header_example1_test</code>	Uses the online example to ensure the correctness of the compression. [41]
<code>fragmentation_test</code>	Tests the fragmentation of an IPv6 packet into smaller 6LoWPAN fragments, then reassembles them to ensure the fragments correctly reconstruct the original packet.
<code>reassemble_full_ipv6_pkt_test</code>	Tests the reassembly process for a fully fragmented IPv6 packet, ensuring all fragments are stored, processed, and reconstructed into the original packet.
<code>extended_EUI64_from_64mac</code>	Verifies the correct handling of converting a 64-bit MAC address to a standardized EUI64 format, maintaining consistency with the 6LoWPAN requirements.
<code>extended_EUI64_from_16mac</code>	Ensures that a 16-bit short MAC address is correctly converted into a EUI64 address.
<code>multicast_addr_validity</code>	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_pkt_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_pkt_comp	<=	0.000s	Ok
9	lowpan_test_SUITE	multicast_addr_pkt_comp	<=	0.000s	Ok
10	lowpan_test_SUITE	global_context_pkt_comp1	<=	0.000s	Ok
11	lowpan_test_SUITE	udp_nh_pkt_comp	<=	0.000s	Ok
12	lowpan_test_SUITE	tcp_nh_pkt_comp	<=	0.000s	Ok
13	lowpan_test_SUITE	icmp_nh_pkt_comp	<=	0.000s	Ok
14	lowpan_test_SUITE	unc_ipv6	<=	0.000s	Ok
15	lowpan_test_SUITE	iphc_pkt_64bit_addr	<=	0.000s	Ok
16	lowpan_test_SUITE	iphc_pkt_16bit_addr	<=	0.000s	Ok
17	lowpan_test_SUITE	msh_pkt	<=	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_pkt	<=	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.

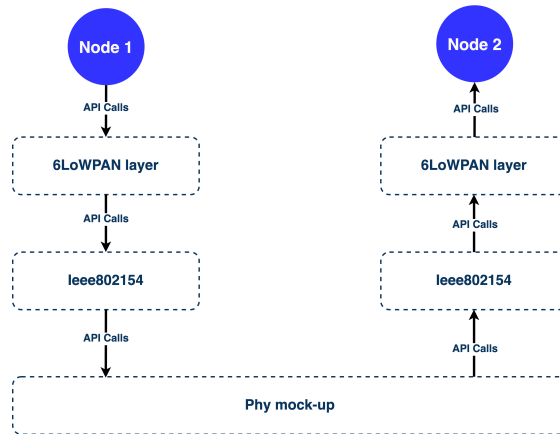


Figure 7.2: Software test workflow

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:C8:FE:DE:CA:00:00:00:01"
Destination address: "FE:80:00:00:00:00:00: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

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

```

Initialization
Current node address: <<200,254,222,202,0,0,0,1>>
"node3MAlr-m1": Routing table server successfully launched
"node3MAlr-m1": IEEE 802.15.4: layer successfully launched
"node3MAlr-m1": 6Loapan Layer successfully launched

Transmission request
Final destination: <<200,254,222,202,0,0,0,2>>
Searching next hop...
Next hop found: <<200,254,222,202,0,0,0,3>>
No fragmentation needed
Packet successfully sent

*** User 2004-08-12 15:47:27.177 ***
Routed packet sent successfully from node1 to node2

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 : <<200,254,222,202,0,0,0,3>>
The datagram needs to be meshed
Searching next hop in the routing table...
Direct link found
Forwarding to node: <<200,254,222,202,0,0,0,2>>

Packet sent successfully

loapan layer stopped

Initialization
Current node address: <<200,254,222,202,0,0,0,2>>
"node2MAlr-m1": Routing table server successfully launched
"node2MAlr-m1": IEEE 802.15.4: layer successfully launched
"node2MAlr-m1": 6Loapan Layer successfully launched

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 loapan 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:C8:FE:0E:CA:00:00:00:01"
Destination address: "FE:80:00:00:00:00:00:C8:FE:0E:CA:00:00:00:02"
Data: <<"Hello world this is an ipv6 packet for testing purpose">>

*** User 2004-08-12 15:47:27.183 ***
Routed packet received successfully at node2

```

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
```

```
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 7.9: Timeout scenario Sender

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

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

DatagramMap after update:
{<<200,254,222,202,0,0,1>>,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.

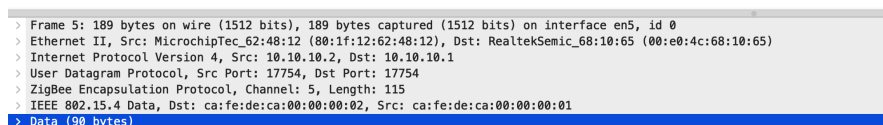
Channel	PRF	Preamble length	Data rate	Preamble code
5	16MHz	1024	6.8 Mbps	4

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



```
> Frame 5: 189 bytes on wire (1512 bits), 189 bytes captured (1512 bits) on interface en5, id 0
> Ethernet II, Src: MicrochipTec_62:48:12 (80:1f:12:62:48:12), Dst: RealtekSemic_68:10:65 (00:e0:4c:68:10:65)
> Internet Protocol Version 4, Src: 10.10.10.2, Dst: 10.10.10.1
> User Datagram Protocol, Src Port: 17754, Dst Port: 17754
> ZigBee Encapsulation Protocol, Channel: 5, Length: 115
> IEEE 802.15.4 Data, Dst: ca:fe:de:ca:00:00:00:02, Src: ca:fe:de:ca:00:00:00:01
> Data (90 bytes)
```

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.

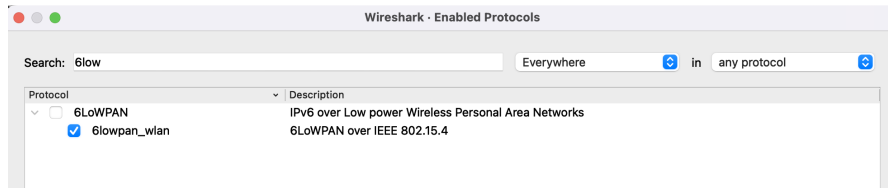


Figure 7.14: 6LoWPAN wireshark activation

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

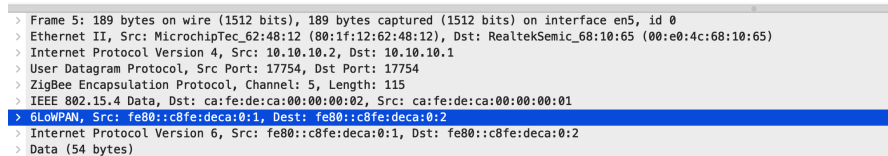


Figure 7.15: Wireshark setup 2

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.

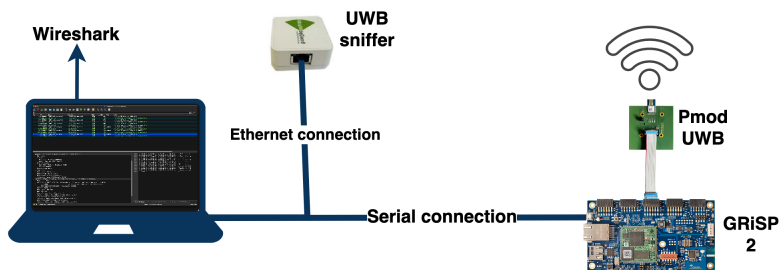


Figure 7.16: Hardware test workflow

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

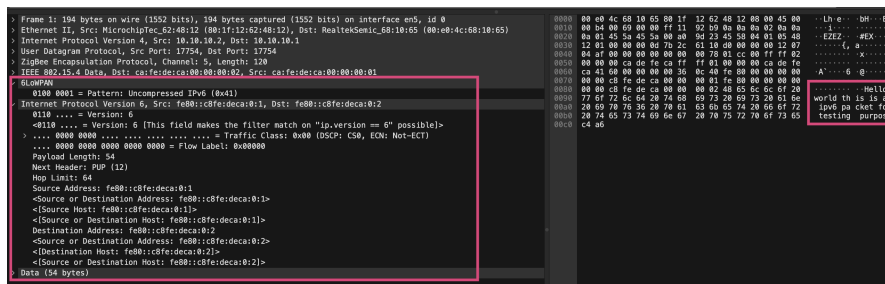


Figure 7.17: Uncompressed IPv6 datagram encapsulation, Wireshark pcap

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

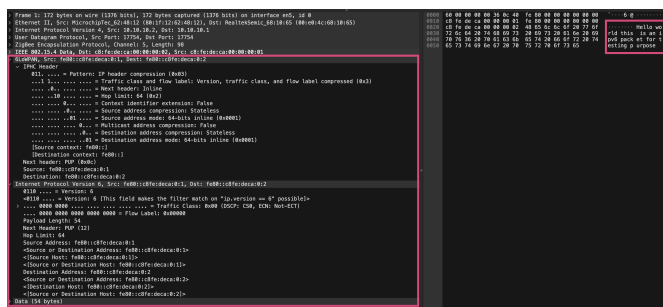


Figure 7.18: Compressed IPv6 datagram encapsulation, Wireshark pcap

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

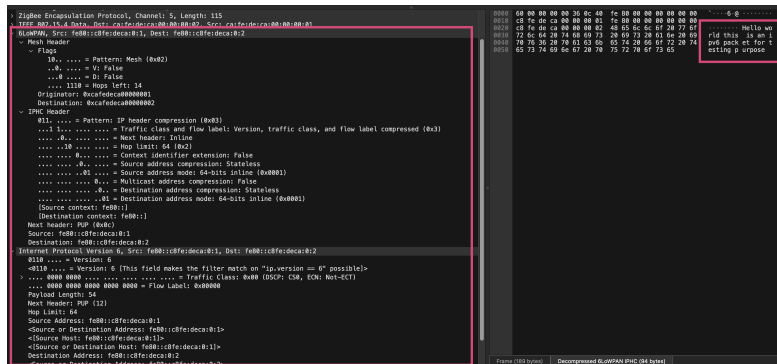


Figure 7.19: Meshed IPv6 datagram encapsulation, Wireshark pcap

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

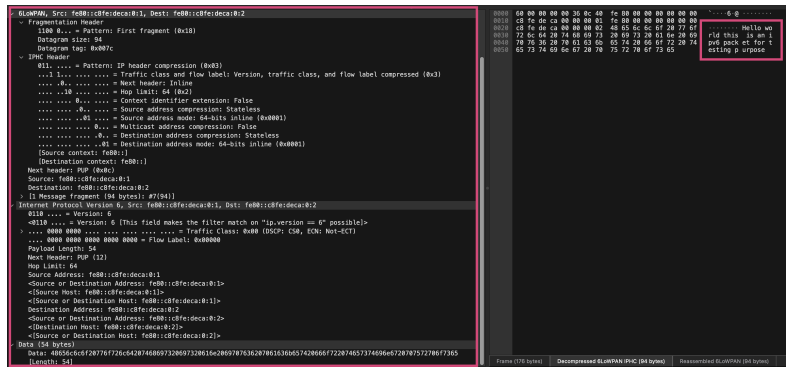


Figure 7.20: Fragmented IPv6 datagram encapsulation, Wireshark pcap

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

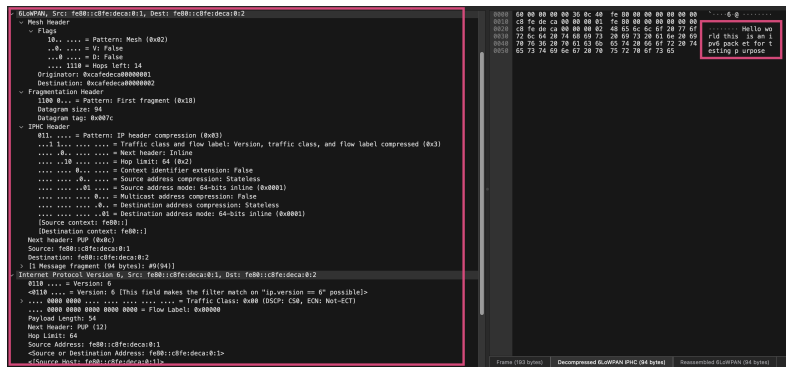


Figure 7.21: Compressed, meshed and fragmented encapsulation, Wireshark pcap

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

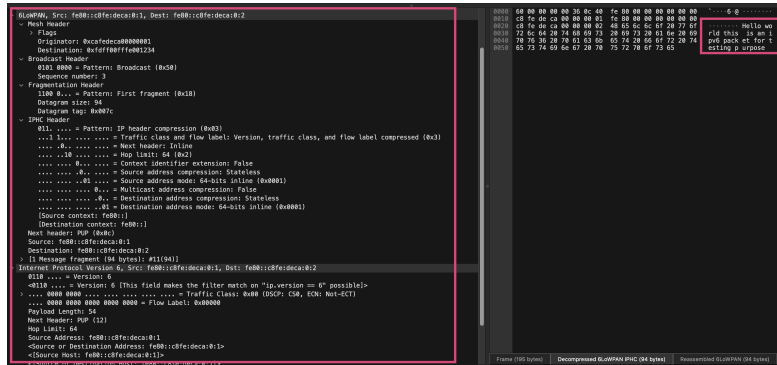


Figure 7.22: Broadcasted IPv6 datagram encapsulation, Wireshark pcap

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

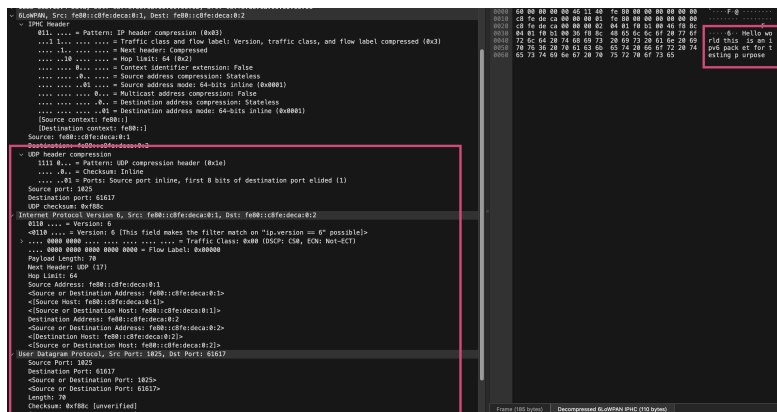


Figure 7.23: UDP encapsulation, Wireshark pcap

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

```

Initialization
Current node address: <<200,254,222,202,0,0,1>>
nonode@nohost: Routing table server successfully launched
nonode@nohost: IEEE 802.15.4 layer successfully launched
nonode@nohost: 6lowpan layer successfully launched

-----

Eshell V14.2.5.1 (press Ctrl+G to abort, type help(). for help)
1> robot:tx().
robot:tx().
Transmission request
Final destination: <<200,254,222,202,0,0,2>>
Searching next hop...
Direct link found
No fragmentation needed
73-byte packet successfully sent

```

Figure 7.24: Simple communication GRiSP Sender

```

Initialization
Current node address: <<200,254,222,202,0,0,2>>
nonode@nohost: Routing table server successfully launched
nonode@nohost: IEEE 802.15.4 layer successfully launched
nonode@nohost: 6lowpan layer successfully launched

-----

Reception mode
New frame received
Originator      : <<200,254,222,202,0,0,1>>
Final destination address: <<200,254,222,202,0,0,2>>
Current node address  : <<200,254,222,202,0,0,2>>
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:C8:FE:DE:CA:00:00:00:01"
Destination address: "FE:80:00:00:00:00:00:C8:FE:DE:CA:00:00:00:02"
Data: <<"Hello world this is an ipv6 packet for testing purpose">>

```

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

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

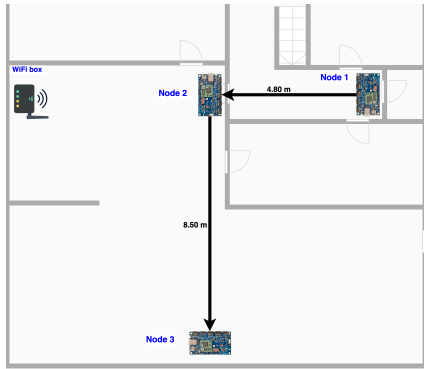


Figure 7.28: Three nodes routing test Sender

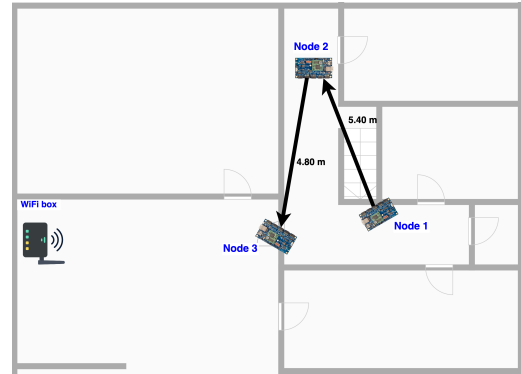


Figure 7.29: Three nodes routing test Receiver

Results

```

9* robottx_big_payload(20).
root@tx_big_payload(20).
Transmission request
Final destination: <<200,254,222,202,0,0,3>>
Searching next hop...
Next hop found: <<200,254,222,202,0,0,2>>
The received IPv6 packet needs fragmentation to be transmitted
11th fragment: 79 bytes sent
20th fragment: 80 bytes sent
3th fragment: 80 bytes sent
4th fragment: 80 bytes sent
5th fragment: 80 bytes sent
6th fragment: 80 bytes sent
7th fragment: 80 bytes sent
8th fragment: 80 bytes sent
9th fragment: 80 bytes sent
10th fragment: 80 bytes sent
11th fragment: 80 bytes sent
12th fragment: 80 bytes sent
13th fragment: 80 bytes sent
14th fragment: 80 bytes sent
15th fragment: 80 bytes sent
16th fragment: 80 bytes sent
17th fragment: 57 bytes sent
Packet successfully sent
  
```

Figure 7.30: Routing test Sender

```

Initialization
Current node address: <<200,254,222,202,0,0,2>>
nonode@nohost: Routing table server successfully launched
nonode@nohost: IEEE 802.15.4 layer successfully launched
nonode@nohost: 6Lowpan layer successfully launched

Reception mode
New frame received
Originator          : <<200,254,222,202,0,0,1>>
Final destination address: <<200,254,222,202,0,0,3>>
Current node address  : <<200,254,222,202,0,0,2>>
The datagram needs to be meshed
Searching next hop in the routing table...
Direct link found
Forwarding to node: <<200,254,222,202,0,0,3>>

New frame received
Originator          : <<200,254,222,202,0,0,1>>
Final destination address: <<200,254,222,202,0,0,3>>
Current node address  : <<200,254,222,202,0,0,2>>
The datagram needs to be meshed
Packet sent successfully
  
```

Figure 7.31: Routing test Forwarder

```

**struct_0*****
  4 *
  5 *
  6 *
  7 *
  8 *
  9 *
 10 *
 11 *
 12 *
 13 *
 14 *
 15 *
 16 *
 17 *
 18 *
 19 *
 20 *
 21 *
 22 *
 23 *
 24 *
 25 *
 26 *
 27 *
 28 *
 29 *
 30 *
 31 *
 32 *
 33 *
 34 *
 35 *
 36 *
 37 *
 38 *
 39 *
 40 *
 41 *
 42 *
 43 *
 44 *
 45 *
 46 *
 47 *
 48 *
 49 *
 50 *
 51 *
 52 *
 53 *
 54 *
 55 *
 56 *
 57 *
 58 *
 59 *
 60 *
 61 *
 62 *
 63 *
 64 *
 65 *
 66 *
 67 *
 68 *
 69 *
 70 *
 71 *
 72 *
 73 *
 74 *
 75 *
 76 *
 77 *
 78 *
 79 *
 80 *
 81 *
 82 *
 83 *
 84 *
 85 *
 86 *
 87 *
 88 *
 89 *
 90 *
 91 *
 92 *
 93 *
 94 *
 95 *
 96 *
 97 *
 98 *
 99 *
100 *
101 *
102 *
103 *
104 *
105 *
106 *
107 *
108 *
109 *
110 *
111 *
112 *
113 *
114 *
115 *
116 *
117 *
118 *
119 *
120 *
121 *
122 *
123 *
124 *
125 *
126 *
127 *
128 *
129 *
130 *
131 *
132 *
133 *
134 *
135 *
136 *
137 *
138 *
139 *
140 *
141 *
142 *
143 *
144 *
145 *
146 *
147 *
148 *
149 *
150 *
151 *
152 *
153 *
154 *
155 *
156 *
157 *
158 *
159 *
160 *
161 *
162 *
163 *
164 *
165 *
166 *
167 *
168 *
169 *
170 *
171 *
172 *
173 *
174 *
175 *
176 *
177 *
178 *
179 *
180 *
181 *
182 *
183 *
184 *
185 *
186 *
187 *
188 *
189 *
190 *
191 *
192 *
193 *
194 *
195 *
196 *
197 *
198 *
199 *
200 *
201 *
202 *
203 *
204 *
205 *
206 *
207 *
208 *
209 *
210 *
211 *
212 *
213 *
214 *
215 *
216 *
217 *
218 *
219 *
220 *
221 *
222 *
223 *
224 *
225 *
226 *
227 *
228 *
229 *
230 *
231 *
232 *
233 *
234 *
235 *
236 *
237 *
238 *
239 *
240 *
241 *
242 *
243 *
244 *
245 *
246 *
247 *
248 *
249 *
250 *
251 *
252 *
253 *
254 *
255 *
256 *
257 *
258 *
259 *
260 *
261 *
262 *
263 *
264 *
265 *
266 *
267 *
268 *
269 *
270 *
271 *
272 *
273 *
274 *
275 *
276 *
277 *
278 *
279 *
280 *
281 *
282 *
283 *
284 *
285 *
286 *
287 *
288 *
289 *
290 *
291 *
292 *
293 *
294 *
295 *
296 *
297 *
298 *
299 *
300 *
301 *
302 *
303 *
304 *
305 *
306 *
307 *
308 *
309 *
310 *
311 *
312 *
313 *
314 *
315 *
316 *
317 *
318 *
319 *
320 *
321 *
322 *
323 *
324 *
325 *
326 *
327 *
328 *
329 *
330 *
331 *
332 *
333 *
334 *
335 *
336 *
337 *
338 *
339 *
340 *
341 *
342 *
343 *
344 *
345 *
346 *
347 *
348 *
349 *
350 *
351 *
352 *
353 *
354 *
355 *
356 *
357 *
358 *
359 *
360 *
361 *
362 *
363 *
364 *
365 *
366 *
367 *
368 *
369 *
370 *
371 *
372 *
373 *
374 *
375 *
376 *
377 *
378 *
379 *
380 *
381 *
382 *
383 *
384 *
385 *
386 *
387 *
388 *
389 *
390 *
391 *
392 *
393 *
394 *
395 *
396 *
397 *
398 *
399 *
400 *
401 *
402 *
403 *
404 *
405 *
406 *
407 *
408 *
409 *
410 *
411 *
412 *
413 *
414 *
415 *
416 *
417 *
418 *
419 *
420 *
421 *
422 *
423 *
424 *
425 *
426 *
427 *
428 *
429 *
430 *
431 *
432 *
433 *
434 *
435 *
436 *
437 *
438 *
439 *
440 *
441 *
442 *
443 *
444 *
445 *
446 *
447 *
448 *
449 *
450 *
451 *
452 *
453 *
454 *
455 *
456 *
457 *
458 *
459 *
460 *
461 *
462 *
463 *
464 *
465 *
466 *
467 *
468 *
469 *
470 *
471 *
472 *
473 *
474 *
475 *
476 *
477 *
478 *
479 *
480 *
481 *
482 *
483 *
484 *
485 *
486 *
487 *
488 *
489 *
490 *
491 *
492 *
493 *
494 *
495 *
496 *
497 *
498 *
499 *
500 *
501 *
502 *
503 *
504 *
505 *
506 *
507 *
508 *
509 *
510 *
511 *
512 *
513 *
514 *
515 *
516 *
517 *
518 *
519 *
520 *
521 *
522 *
523 *
524 *
525 *
526 *
527 *
528 *
529 *
530 *
531 *
532 *
533 *
534 *
535 *
536 *
537 *
538 *
539 *
540 *
541 *
542 *
543 *
544 *
545 *
546 *
547 *
548 *
549 *
550 *
551 *
552 *
553 *
554 *
555 *
556 *
557 *
558 *
559 *
560 *
561 *
562 *
563 *
564 *
565 *
566 *
567 *
568 *
569 *
570 *
571 *
572 *
573 *
574 *
575 *
576 *
577 *
578 *
579 *
580 *
581 *
582 *
583 *
584 *
585 *
586 *
587 *
588 *
589 *
590 *
591 *
592 *
593 *
594 *
595 *
596 *
597 *
598 *
599 *
600 *
601 *
602 *
603 *
604 *
605 *
606 *
607 *
608 *
609 *
610 *
611 *
612 *
613 *
614 *
615 *
616 *
617 *
618 *
619 *
620 *
621 *
622 *
623 *
624 *
625 *
626 *
627 *
628 *
629 *
630 *
631 *
632 *
633 *
634 *
635 *
636 *
637 *
638 *
639 *
640 *
641 *
642 *
643 *
644 *
645 *
646 *
647 *
648 *
649 *
650 *
651 *
652 *
653 *
654 *
655 *
656 *
657 *
658 *
659 *
660 *
661 *
662 *
663 *
664 *
665 *
666 *
667 *
668 *
669 *
670 *
671 *
672 *
673 *
674 *
675 *
676 *
677 *
678 *
679 *
680 *
681 *
682 *
683 *
684 *
685 *
686 *
687 *
688 *
689 *
690 *
691 *
692 *
693 *
694 *
695 *
696 *
697 *
698 *
699 *
700 *
701 *
702 *
703 *
704 *
705 *
706 *
707 *
708 *
709 *
710 *
711 *
712 *
713 *
714 *
715 *
716 *
717 *
718 *
719 *
720 *
721 *
722 *
723 *
724 *
725 *
726 *
727 *
728 *
729 *
730 *
731 *
732 *
733 *
734 *
735 *
736 *
737 *
738 *
739 *
740 *
741 *
742 *
743 *
744 *
745 *
746 *
747 *
748 *
749 *
750 *
751 *
752 *
753 *
754 *
755 *
756 *
757 *
758 *
759 *
760 *
761 *
762 *
763 *
764 *
765 *
766 *
767 *
768 *
769 *
770 *
771 *
772 *
773 *
774 *
775 *
776 *
777 *
778 *
779 *
780 *
781 *
782 *
783 *
784 *
785 *
786 *
787 *
788 *
789 *
790 *
791 *
792 *
793 *
794 *
795 *
796 *
797 *
798 *
799 *
800 *
801 *
802 *
803 *
804 *
805 *
806 *
807 *
808 *
809 *
810 *
811 *
812 *
813 *
814 *
815 *
816 *
817 *
818 *
819 *
820 *
821 *
822 *
823 *
824 *
825 *
826 *
827 *
828 *
829 *
830 *
831 *
832 *
833 *
834 *
835 *
836 *
837 *
838 *
839 *
840 *
841 *
842 *
843 *
844 *
845 *
846 *
847 *
848 *
849 *
850 *
851 *
852 *
853 *
854 *
855 *
856 *
857 *
858 *
859 *
860 *
861 *
862 *
863 *
864 *
865 *
866 *
867 *
868 *
869 *
870 *
871 *
872 *
873 *
874 *
875 *
876 *
877 *
878 *
879 *
880 *
881 *
882 *
883 *
884 *
885 *
886 *
887 *
888 *
889 *
890 *
891 *
892 *
893 *
894 *
895 *
896 *
897 *
898 *
899 *
900 *
901 *
902 *
903 *
904 *
905 *
906 *
907 *
908 *
909 *
910 *
911 *
912 *
913 *
914 *
915 *
916 *
917 *
918 *
919 *
920 *
921 *
922 *
923 *
924 *
925 *
926 *
927 *
928 *
929 *
930 *
931 *
932 *
933 *
934 *
935 *
936 *
937 *
938 *
939 *
940 *
941 *
942 *
943 *
944 *
945 *
946 *
947 *
948 *
949 *
950 *
951 *
952 *
953 *
954 *
955 *
956 *
957 *
958 *
959 *
960 *
961 *
962 *
963 *
964 *
965 *
966 *
967 *
968 *
969 *
970 *
971 *
972 *
973 *
974 *
975 *
976 *
977 *
978 *
979 *
980 *
981 *
982 *
983 *
984 *
985 *
986 *
987 *
988 *
989 *
990 *
991 *
992 *
993 *
994 *
995 *
996 *
997 *
998 *
999 *
1000 *
1001 *
1002 *
1003 *
1004 *
1005 *
1006 *
1007 *
1008 *
1009 *
1010 *
1011 *
1012 *
1013 *
1014 *
1015 *
1016 *
1017 *
1018 *
1019 *
1020 *
1021 *
1022 *
1023 *
1024 *
1025 *
1026 *
1027 *
1028 *
1029 *
1030 *
1031 *
1032 *
1033 *
1034 *
1035 *
1036 *
1037 *
1038 *
1039 *
1040 *
1041 *
1042 *
1043 *
1044 *
1045 *
1046 *
1047 *
1048 *
1049 *
1050 *
1051 *
1052 *
1053 *
1054 *
1055 *
1056 *
1057 *
1058 *
1059 *
1060 *
1061 *
1062 *
1063 *
1064 *
1065 *
1066 *
1067 *
1068 *
1069 *
1070 *
1071 *
1072 *
1073 *
1074 *
1075 *
1076 *
1077 *
1078 *
1079 *
1080 *
1081 *
1082 *
1083 *
1084 *
1085 *
1086 *
1087 *
1088 *
1089 *
1090 *
1091 *
1092 *
1093 *
1094 *
1095 *
1096 *
1097 *
1098 *
1099 *
1100 *
1101 *
1102 *
1103 *
1104 *
1105 *
1106 *
1107 *
1108 *
1109 *
1110 *
1111 *
1112 *
1113 *
1114 *
1115 *
1116 *
1117 *
1118 *
1119 *
1120 *
1121 *
1122 *
1123 *
1124 *
1125 *
1126 *
1127 *
1128 *
1129 *
1130 *
1131 *
1132 *
1133 *
1134 *
1135 *
1136 *
1137 *
1138 *
1139 *
1140 *
1141 *
1142 *
1143 *
1144 *
1145 *
1146 *
1147 *
1148 *
1149 *
1150 *
1151 *
1152 *
1153 *
1154 *
1155 *
1156 *
1157 *
1158 *
1159 *
1160 *
1161 *
1162 *
1163 *
1164 *
1165 *
1166 *
1167 *
1168 *
1169 *
1170 *
1171 *
1172 *
1173 *
1174 *
1175 *
1176 *
1177 *
1178 *
1179 *
1180 *
1181 *
1182 *
1183 *
1184 *
1185 *
1186 *
1187 *
1188 *
1189 *
1190 *
1191 *
1192 *
1193 *
1194 *
1195 *
1196 *
1197 *
1198 *
1199 *
1200 *
1201 *
1202 *
1203 *
1204 *
1205 *
1206 *
1207 *
1208 *
1209 *
1210 *
1211 *
1212 *
1213 *
1214 *
1215 *
1216 *
1217 *
1218 *
1219 *
1220 *
1221 *
1222 *
1223 *
1224 *
1225 *
1226 *
1227 *
1228 *
1229 *
1230 *
1231 *
1232 *
1233 *
1234 *
1235 *
1236 *
1237 *
1238 *
1239 *
1240 *
1241 *
1242 *
1243 *
1244 *
1245 *
1246 *
1247 *
1248 *
1249 *
1250 *
1251 *
1252 *
1253 *
1254 *
1255 *
1256 *
1257 *
1258 *
1259 *
1260 *
1261 *
1262 *
1263 *
1264 *
1265 *
1266 *
1267 *
1268 *
1269 *
1270 *
1271 *
1272 *
1273 *
1274 *
1275 *
1276 *
1277 *
1278 *
1279 *
1280 *
1281 *
1282 *
1283 *
1284 *
1285 *
1286 *
1287 *
1288 *
1289 *
1290 *
1291 *
1292 *
1293 *
1294 *
1295 *
1296 *
1297 *
1298 *
1299 *
1300 *
1301 *
1302 *
1303 *
1304 *
1305 *
1306 *
1307 *
1308 *
1309 *
1310 *
1311 *
1312 *
1313 *
1314 *
1315 *
1316 *
1317 *
1318 *
1319 *
1320 *
1321 *
1322 *
1323 *
1324 *
1325 *
1326 *
1327 *
1328 *
1329 *
1330 *
1331 *
1332 *
1333 *
1334 *
1335 *
1336 *
1337 *
1338 *
1339 *
1340 *
1341 *
1342 *
1343 *
1344 *
1345 *
1346 *
1347 *
1348 *
1349 *
1350 *
1351 *
1352 *
1353 *
1354 *
1355 *
1356 *
1357 *
1358 *
1359 *
1360 *
1361 *
1362 *
1363 *
1364 *
1365 *
1366 *
1367 *
1368 *
1369 *
1370 *
1371 *
1372 *
1373 *
1374 *
1375 *
1376 *
1377 *
1378 *
1379 *
1380 *
1381 *
1382 *
1383 *
1384 *
1385 *
1386 *
1387 *
1388 *
1389 *
1390 *
1391 *
1392 *
1393 *
1394 *
1395 *
1396 *
1397 *
1398 *
1399 *
1400 *
1401 *
1402 *
1403 *
1404 *
1405 *
1406 *
1407 *
1408 *
1409 *
1410 *
1411 *
1412 *
1413 *
1414 *
1415 *
1416 *
1417 *
1418 *
1419 *
1420 *
1421 *
1422 *
1423 *
1424 *
1425 *
1426 *
1427 *
1428 *
1429 *
1430 *
1431 *
1432 *
1433 *
1434 *
1435 *
1436 *
1437 *
1438 *
1439 *
1440 *
1441 *
1442 *
1443 *
1444 *
1445 *
1446 *
1447 *
1448 *
1449 *
1450 *
1451 *
1452 *
1453 *
1454 *
1455 *
1456 *
1457 *
1458 *
1459 *
1460 *
1461 *
1462 *
1463 *
1464 *
1465 *
1466 *
1467 *
1468 *
1469 *
1470 *
1471 *
1472 *
1473 *
1474 *
1475 *
1476 *
1477 *
1478 *
1479 *
1480 *
1481 *
1482 *
1483 *
1484 *
1485 *
1486 *
1487 *
1488 *
1489 *
1490 *
1491 *
1492 *
1493 *
1494 *
1495 *
1496 *
1497 *
1498 *
1499 *
1500 *
1501 *
1502 *
1503 *
1504 *
1505 *
1506 *
1507 *
1508 *
1509 *
1510 *
1511 *
1512 *
1513 *
1514 *
1515 *
1516 *
1517 *
1518 *
1519 *
1520 *
1521 *
1522 *
1523 *
1524 *
1525 *
1526 *
1527 *
1528 *
1529 *
1530 *
1531 *
1532 *
1533 *
1534 *
1535 *
1536 *
1537 *
1538 *
1539 *
1540 *
1541 *
1542 *
1543 *
1544 *
1545 *
1546 *
1547 *
1548 *
1549 *
1550 *
1551 *
1552 *
1553 *
1554 *
1555 *
1556 *
1557 *
1558 *
1559 *
1560 *
1561 *
1562 *
1563 *
1564 *
1565 *
1566 *
1567 *
1568 *
1569 *
1570 *
1571 *
1572 *
1573 *
1574 *
1575 *
1576 *
1577 *
1578 *
1579 *
1580 *
1581 *
1582 *
1583 *
1584 *
1585 *
1586 *
1587 *
1588 *
1589 *
1590 *
1591 *
1592 *
1593 *
1594 *
1595 *
1596 *
1597 *
1598 *
1599 *
1600 *
1601 *
1602 *
1603 *
1604 *
1605 *
1606 *
1607 *
1608 *
1609 *
1610 *
1611 *
1612 *
1613 *
1614 *
1615 *
1616 *
1617 *
1618 *
1619 *
1620 *
1621 *
1622 *
1623 *
1624 *
1625 *
1626 *
1627 *
1628 *
1629 *
1630 *
1631 *
1632 *
1633 *
1634 *
1635 *
1636 *
1637 *
1638 *
1639 *
1640 *
1641 *
1642 *
1643 *
1644 *
1645 *
1646 *
1647 *
1648 *
1649 *
1650 *
1651 *
1652 *
1653 *
1654 *
1655 *
1656 *
1657 *
1658 *
1659 *
1660 *
1661 *
1662 *
1663 *
1664 *
1665 *
1666 *
1667 *
1668 *
1669 *
1670 *
1671 *
1672 *
1673 *
1674 *
1675 *
1676 *
1677 *
1678 *
1679 *
1680 *
1681 *
1682 *
1683 *
1684 *
1685 *
1686 *
1687 *
1688 *
1689 *
1690 *
1691 *
1692 *
1693 *
1694 *
1695 *
1696 *
1697 *
1698 *
1699 *
1700 *
1701 *
1702 *
1703 *
1704 *
1705 *
1706 *
1707 *
1708 *
1709 *
1710 *
1711 *
1712 *
1713 *
1714 *
1715 *
1716 *
1717 *
1718 *
1719 *
1720 *
1721 *
1722 *
1723 *
1724 *
1725 *
1726 *
1727 *
1728 *
1729 *
1730 *
1731 *
1732 *
1733 *
1734 *
1735 *
1736 *
1737 *
1738 *
1739 *
1740 *
1741 *
1742 *
1743 *
1744 *
1745 *
1746 *
1747 *
1748 *
1749 *
1750 *
1751 *
1752 *
1753 *
1754 *
1755 *
1756 *
1757 *
1758 *
1759 *
1760 *
1761 *
1762 *
1763 *
1764 *
1765 *
1766 *
1767 *
1768 *
1769 *
1770 *
1771 *
1772 *
1773 *
1774 *
1775 *
1776 *
1777 *
1778 *
1779 *
1780 *
1781 *
1782 *
1783 *
1784 *
1785 *
1786 *
1787 *
1788 *
1789 *
1790 *
1791 *
1792 *
1793 *
1794 *
1795 *
1796 *
1797 *
1798 *
1799 *
1800 *
1801 *
1802 *
1803 *
1804 *
1805 *
1806 *
1807 *
1808 *
1809 *
1810 *
1811 *
1812 *
1813 *
1814 *
1815 *
1816 *
1817 *
1818 *
1819 *
1820 *
1821 *
1822 *
1823 *
1824 *
1825 *
1826 *
1827 *
1828 *
1829 *
1830 *
1831 *
1832 *
1833 *
1834 *
1835 *
1836 *
1837 *
1838 *
1839 *
1840 *
1841 *
1842 *
1843 *
1844 *
1845 *
1846 *
1847 *
1848 *
1849 *
1850 *
1851 *
1852 *
1853 *
1854 *
1855 *
1856 *
1857 *
1858 *
1859 *
1860 *
1861 *
1862 *
1863 *
1864 *
1865 *
1866 *
1867 *
1868 *
1869 *
1870 *
1871 *
1872 *
1873 *
1874 *
1875 *
1876 *
1877 *
1878 *
1879 *
1880 *
1881 *
1882 *
1883 *
1884 *
1885 *
1886 *
1887 *
1888 *
1889 *
1890 *
1891 *
1892 *
1893 *
1894 *
1895 *
1896 *
1897 *
1898 *
1899 *
1900 *
1901 *
1902 *
1903 *
1904 *
1905 *
1906 *
1907 *
1908 *
1909 *
1910 *
1911 *
1912 *
1913 *
1914 *
1915 *
1916 *
1917 *
1918 *
1919 *
1920 *
1921 *
1922 *
1923 *
1924 *
1925 *
1926 *
1927 *
1928 *
1929 *
1930 *
1931 *
1932 *
1933 *
1934 *
1935 *
1936 *
1937 *
1938 *
1939 *
1940 *
1941 *
1942 *
1943 *
1944 *
1945 *
1946 *
1947 *
1948 *
1949 *
1950 *
1951 *
1952 *
1953 *
1
```

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.

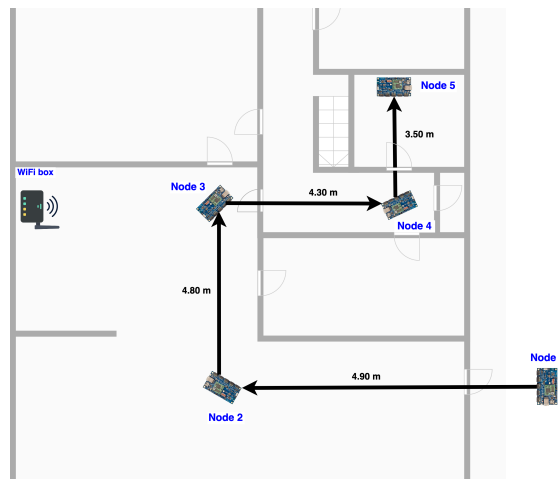


Figure 7.33: Five nodes routing test

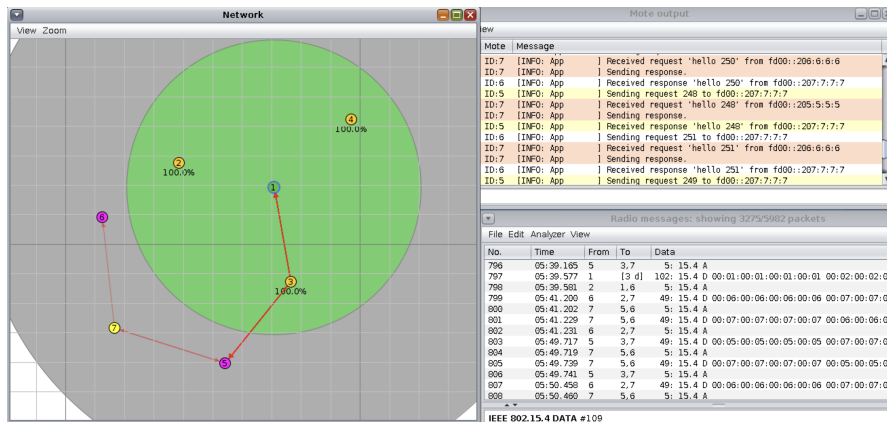


Figure 7.37: UDP border router cooja simulation

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

```

- ICMP Header
  #11 ..... = Pattern: IP header compression (0x03)
  ...1 ..... = Traffic class and flow label: Version, traffic class, and flow label compressed (0x3)
  .... 00 ..... = Next Header: 0x58
  .... ..18 ..... = Hop Limit: 64 (0x2)
  .... ..00 ..... = Context identifier extension: False
  .... ..00 ..... = Source address compression: Stateless
  .... ..11 ..... = Source address mode: Compressed (0x0003)
  .... ..00 ..... = Multicast address compression: True
  .... ..00 ..... = Destination address compression: Stateless
  .... ..13 ..... = Destination address mode: 8-bits in-line (0x0003)
  [Source context: fe80:]
  [Destination context: fe02:]
  Next header: ICMPv6 (0x3a)
  [Source: fe80::207:7:7:7]
  [Destination: ff02::1a]
Internet Protocol, Version 6, Src: fe80::207:7:7:7, Dst: ff02::1a
#118 ..... = Version: 6
<#118 ..... = Version: 6 This field makes the filter match on "ip.version eq 6" possible!>
..... 0000 0000 ..... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-Set)
..... 0000 0000 0000 0000 ..... = Flow Label: 0x000000
Payload Length: 0
Next Header: ICMPv6 (58)
Hop Limit: 64
Source Address: fe80::207:7:7:7

```

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

```

0x0000, Src: fe80::202:2:2:2, Dest: fe80::212:7402:2:202
- IPHC header
  #11 ... = Pattern: IP header compression (0x03)
  ... 1 ... = Traffic class and flow labels, version, traffic class, and flow label compressed (0x3)
  ... 1 ... = Next header: Compressed
  ... 18 ... = Hop limit: 64 (0x2)
  ... 0 ... = Context identifier extension: False
  ... 0 ... = Source address compression: Stateless
  ... 11 ... = Source address mode: Compressed (0x0003)
  ... 0 ... = Multicast address compression: False
  ... 0 ... = Destination address compression: Stateless
  ... 11 ... = Destination address mode: Compressed (0x0003)
  [Source context: fe80::]
  [Destination context: fe80::]
  Source: fe80::202:2:2:2
  Destination: fe80::212:7402:2:202
- UDP header (compression)
  #11 0 ... = Pattern: UDP compression header (0x1e)
  ... 0 ... = Checksum: In-line
  ... 08 ... = Ports: In-line (0)
  Source port: 5683
  Destination port: 5683
  UDP checksum: 8441
Internet Protocol Version 6, Src: fe80::202:2:2:2, Dest: fe80::212:7402:2:202
#19 ... = Version: 6
  
```

Figure 7.40: Contiki 2nd example Wireshark capture

```

Expected carried values: #{}
Actual carried values: #{"DAM" => <<0,212,28,234,0,2,0,2>>,
                      "SAM" => <<0,202,0,2,0,2,0,2>>}
  
```

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

```

ELENPWA, Src: 1287:7:7:7, Dest: 1280:2:2:2
- IPv6 header
  011 ... = Pattern: IP header compression (8x03)
  ...1 ... = Traffic class and flow labels: Version, traffic class, and flow label compressed (8x3)
  ...1 ... = Next header: Compressed
  ...00 ... = Hop limit: In-line (0x0)
  ...00 ... = Context identifier: extension: True
  ...1 ... = Source address compression: Stateful
  ...00 ... = Source address mode: 64-bits in-line (8x0001)
  ...0 ... = Multicast address compression: False
  ...1 ... = Destination address compression: Stateful
  ...11 ... = Destination address mode: Compressed (8x0003)
  0000 ... = Source context identifier: 8x0
  ...0000 ... = Destination context identifier: 8x0
Hop limit: 63
Source: 1287:7:7:7
Destination: 1280:2:2:2
- IPv6 extension header
  118 ... = Pattern: IPv6 extension header (8x0e)
  ...03 ... = Header ID: IPv6 routing (8x3)
  ...0 ... = Next header: In-line
  Next header: ECPv6 (8x0a)
  Header length: 14
  [Data 14 bytes]
Internet Protocol, Version 6, Src: 1287:7:7:7, Dest: 1280:2:2:2

```

Figure 7.42: Contiki 3rd example Wireshark capture

```

Expected carried values: #{"HopLimit" => 63,"NH" => 43,
                        "SAM" => <<0,207,0,7,0,7,0,7>>}

Actual carried values: #{"DAM" => <<0,203,0,3,0,3,0,3>>,
                      "HopLimit" => 63,"NextHeader" => 43,
                      "SAM" => <<0,207,0,7,0,7,0,7>>}

```

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 Thread. *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=0JMVO3HN0xo>.
- [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

```
1 % @doc 6LoWPAN header
2 -include("ieee802154.hrl").
3 -include("mac_frame.hrl").
4
5 %-----
6 % Useful records
7 %-----
8 -record(ipv6PckInfo,
9     {version = 6,
10      trafficClass,
11      flowLabel,
12      payloadLength,
13      nextHeader,
14      hopLimit,
15      sourceAddress,
16      destAddress,
17      payload}).
18 -record(datagramInfo, {fragtype, datagramSize, datagramTag, datagramOffset,
19      payload}).
20
```

```

21 %-----
22 % Dispatch Type and Header
23 %-----
24
25 %@doc dispatch value bit pattern from rfc4944, DH stands for dispatch header
26
27 -define(NALP_DHTYPE, 2#00). % Not a LoWPAN frame, such packet shall be discarded
28 -define(IPV6_DHTYPE, 2#01000001). % Uncompressed IPv6 Addresses
29 -define(IPHC_DHTYPE, 2#011). % LOWPAN_IPHC compressed IPv6 (RFC6282)
30 -define(BCO_DHTYPE, 2#01010000). % LOWPAN_BCO broadcast
31 -define(ESC_DHTYPE, 2#01111111). % Additional Dispatch byte follows
32 -define(MESH_DHTYPE, 2#10). % Mesh Header
33 -define(FRAG1_DHTYPE, 2#11000). % First fragmentation Header
34 -define(FRAGN_DHTYPE, 2#11100). % Subsequent fragmentation Header
35 -define(UDP_DHTYPE, 2#11110). % UDP header compression
36
37 -define(0xf0b, 2#111100001011).
38 -define(0xf0, 2#11110000).
39
40 -type dispatch_type() ::
41     ?NALP_DHTYPE |
42     ?IPV6_DHTYPE |
43     ?IPHC_DHTYPE |
44     ?BCO_DHTYPE |
45     ?ESC_DHTYPE |
46     ?MESH_DHTYPE |
47     ?FRAG1_DHTYPE |
48     ?FRAGN_DHTYPE.
49
50 %-----
51 % Fragmentation Type and Header
52 %-----
53
54 -type frag_type() :: ?FRAG1_DHTYPE | ?FRAGN_DHTYPE.
55
56 -record(frag_header,
57     {frag_type = ?FRAG1_DHTYPE :: frag_type(),
58     datagram_size, % 11 bits
59     datagram_tag, % 16 bits
60     datagram_offset}). % 8-bits
61
62 -record(frag_info,
63     {datagram_size,
64     datagram_tag,
65     datagram_offset
66     }).
67
68 -record(datagram,
69     {tag,
70     size,
71     cmpt,
72     timer,
73     fragments
74     }).
75
76 -define(MAX_FRAME_SIZE,80). % Because IEEE 802.15.4 leaves approximately 80-100
77     bytes of payload
77 -define(MAX_FRAG_SIZE_NoMESH,75). % Because max frame size is 80 bytes, and lowpan
78     header 30 bytes (5 bytes for fragHeader) 8 bytes are from IPHC which is
78     included in payload for frag
78 -define(MAX_FRAG_SIZE_MESH,58). % Considering max frame size = 80 bytes, lowpan
79     header = 30 bytes (17 bytes for meshHeader, 5 bytes for fragHeader, 8 bytes

```



```

    for IPHC)
79 -define(MAX_DTG_SIZE, 2047). % 11 bits datagram_size
80 -define(REASSEMBLY_TIMEOUT, 60000). % 60 sec
81 -define(FRAG_HEADER_SIZE,5). % 5 bytes including frag_type, datagram_size,
    datagram_tag, and datagram_offset
82 -define(DATAGRAMS_MAP,#{}). % map of received datagrams, the keys are the tag of
    datagrams
83 -define(MAX_TAG_VALUE, 65535).
84 -define(DEFAULT_TAG_VALUE, 2#0000000000000000).
85 -define(BC_SEQNUM, 2#00000000).
86
87 -record(additional_info,
88     {datagram_size,
89     datagram_tag,
90     hops_left,
91     timer
92     }).
93 -define(INFO_ON, 1).
94 -define(INFO_OFF, 0).
95
96 %-----
97 % Header Compression
98 %-----
99 -record(ipv6_header,
100     {version = 2#0110, % 4-bit
101     traffic_class, % 8-bit
102     flow_label, % 20-bit
103     payload_length, % 16-bit
104     next_header, % 8-bit
105     hop_limit, % 8-bit
106     source_address, % 128-bit
107     destination_address}). % 128-bit
108 -record(udp_header,
109     {source_port, % 16-bit
110     destination_port, % 16-bit
111     length, % 16-bit
112     checksum}). % 16-bit
113 -record(iphc_header,
114     {dispatch = ?IPHC_DHTYPE, % 3-bit dispatch value
115     tf, % 2-bit field for Traffic Class and Flow Control compression options
116     nh, % 1-bit field for Next Header encoding using NHC
117     hlim, % 2-bit field for Hop Limit compression
118     cid, % 1-bit field for Context Identifier Extension
119     sac, % 1-bit field for Source Address Compression (stateless or stateful)
120     sam, % 2-bit field for Source Address Mode
121     m, % 1-bit field for Multicast Compression
122     dac, % 1-bit field for Destination Address Compression (stateless or
        stateful)
123     dam}). % 2-bit field for Destination Address Mode
124
125 -define(LINK_LOCAL_PREFIX, 16#FE80).
126 -define(MULTICAST_PREFIX, 16#FF02).
127 -define(GLOBAL_PREFIX_1, 16#2001).
128 -define(GLOBAL_PREFIX_3, 16#2003).
129 -define(MESH_LOCAL_PREFIX, 16#FD00).
130 -type prefix_type() :: ?LINK_LOCAL_PREFIX | ?GLOBAL_PREFIX_1 | ?MULTICAST_PREFIX.
131
132
133 -define(UDP_PN, 17). % PN stands for Protocol Number
134 -define(TCP_PN, 6).
135 -define(ICMP_PN, 58).
136

```

```

137 % inspired from Thread Usage of 6LoWPAN
138 -define(Context_id_table,
139     #{1 => <<?MESH_LOCAL_PREFIX:16,16#0DB8:16, 0:32>>, % mesh local prefix
140       2 => <<0:64>>, % cooja mesh local prefix
141       %2 => <<?GLOBAL_PREFIX_1:16, 0:48>>, % global prefix 1
142       3 => <<?GLOBAL_PREFIX_3:16, 0:48>>}). % global prefix 3
143
144 -define(Prefix_id_table,
145     #{<<?MESH_LOCAL_PREFIX:16, 0:48>> => 1 , % mesh local prefix
146       <<0:64>> => 2, % cooja mesh local prefix
147       % <<?GLOBAL_PREFIX_1:16, 0:48>> => 2, % global prefix 1
148       <<?GLOBAL_PREFIX_3:16, 0:48>> => 3}). % global prefix 3
149
150 -define(SHORT_ADDR_LEN, 2).
151 -define(EXTENDED_ADDR_LEN, 8).
152
153 %-----
154 % Routing
155 %-----
156
157 -define(BroadcastAdd, <<"      ">>).
158 -define(ACK_FRAME, <<>>).
159
160 -record(mesh_header,
161     {mesh_type = ?MESH_DHTYPE,
162       v_bit, % 1-bit
163       f_bit, % 1-bit
164       hops_left, % 4-bit
165       originator_address, % link-layer address of the Originator
166       final_destination_address % link-layer address of the Final Destination
167       %deep_hops_left = undefined
168     }).
169
170 -record(meshInfo,
171     {version = 6,
172       v_bit,
173       f_bit,
174       hops_left,
175       originator_address,
176       final_destination_address,
177       deep_hops_left,
178       payload}).
179
180
181 -define(Max_Hops, 2#1110).
182 -define(DeepHopsLeft, 16#F). % 0xF
183 -define(Max_DeepHopsLeft, 2#11111111). % 8-bit Deep Hops Left
184
185 -define(Node1MacAddress, <<16#CAFEDECA00000001:64>>).
186 -define(Node2MacAddress, <<16#CAFEDECA00000002:64>>).
187 -define(Node3MacAddress, <<16#CAFEDECA00000003:64>>).
188 -define(Node4MacAddress, <<16#CAFEDECA00000004:64>>).
189 -define(Node5MacAddress, <<16#CAFEDECA00000005:64>>).

```

```

190
191 % Used to test 16-bit node addresses
192 % -define(Node1MacAddress, <<16#0001:16>>).
193 % -define(Node2MacAddress, <<16#0002:16>>).
194 % -define(Node3MacAddress, <<16#0003:16>>).
195
196
197 -define(node1_addr,
198     lowpan_core:generateEUI64MacAddr(?Node1MacAddress)).
199 -define(node2_addr,
200     lowpan_core:generateEUI64MacAddr(?Node2MacAddress)).
201 -define(node3_addr,
202     lowpan_core:generateEUI64MacAddr(?Node3MacAddress)).
203 -define(node4_addr,
204     lowpan_core:generateEUI64MacAddr(?Node4MacAddress)).
205 -define(node5_addr,
206     lowpan_core:generateEUI64MacAddr(?Node5MacAddress)).
207
208 -define(Default_routing_table,
209     #{?node1_addr => ?node1_addr,
210       ?node2_addr => ?node2_addr,
211       ?node3_addr => ?node3_addr,
212       ?node4_addr => ?node4_addr,
213       ?node5_addr => ?node5_addr}).
214
215 % -define(Node1_routing_table,
216 %     #{?node1_addr => ?node1_addr,
217 %       ?node2_addr => ?node3_addr,
218 %       ?node3_addr => ?node2_addr}).
219
220 -define(Node1_routing_table, % 5 node routing test: 1 -> 2 -> 5
221     #{?node5_addr => ?node2_addr,
222       ?node2_addr => ?node2_addr,
223       ?node3_addr => ?node3_addr,
224       ?node4_addr => ?node4_addr}).
225
226 -define(Node2_routing_table, % 5 node routing test: 2 -> 3 -> 5
227     #{?node5_addr => ?node3_addr}).
228
229 -define(Node3_routing_table, % 5 node routing test: 3 -> 4 -> 5
230     #{?node5_addr => ?node4_addr}).
231
232 -define(Node4_routing_table, % 5 node routing test: 4 -> 5
233     #{?node5_addr => ?node5_addr}).
234
235 -define(Node5_routing_table, % 5 node routing test
236     #{?node5_addr => ?node5_addr}).
237
238
239
240
241
242 %-----
243 % Metrics
244 %-----
245 -record(metrics,
246     {ack_counter = 0,
247       fragments_nbr = 1,
248       start_time = 0,
249       end_time = 0,
250       pkt_len = 0,
251       compressed_pkt_len = 0}).

```

A.4 Lowpan core code

```
1 -module(lowpan_core).
2
3 -include("lowpan.hrl").
4
5 -export([
6     pktEncapsulation/2, fragmentIpv6Packet/3,
7     reassemble/1, storeFragment/8, createIphcPckt/2, getIpv6Pkt/2, datagramInfo/1,
8     compressIpv6Header/2, buildDatagramPckt/2, buildFirstFragPckt/5,
9     getPcktInfo/1, getIpv6Payload/1, triggerFragmentation/3,
10    decodeIpv6Pckt/4, encodeInteger/1,
11    tupleToBin/1, buildFragHeader/1, getNextHop/6,
12    generateChunks/0, generateChunks/1,
13    buildMeshHeader/1, getMeshInfo/1, containsMeshHeader/1,
14    buildFirstFragHeader/1, getUncIpv6/1, getEUI64From48bitMac/1, generateLLAddr
15    /1,
16    getEUI64MacAddr/1, createNewMeshHeader/3, createNewMeshDatagram/3,
17    removeMeshHeader/2,
18    convertAddrToBin/1, checkTagUnicity/2, get16bitMacAddr/1, generateMulticastAddr
19    /1,
20    getDecodeIpv6PcktInfo/1, getNextHop/2, generateEUI64MacAddr/1
21 ])
22
23 %% @doc Returns an Ipv6 packet
24 %% @spec getIpv6Pkt(Header, Payload) -> binary().
25
26 -spec getIpv6Pkt(Header, Payload) -> binary() when
27     Header :: binary(),
28     Payload :: binary().
29 getIpv6Pkt(Header, Payload) ->
30     ipv6:buildIpv6Packet(Header, Payload).
31
32 %%
33 %% @doc create an uncompressed IPv6 packet
34 %% @spec pktEncapsulation(Header, Payload) -> binary().
35
36 -spec pktEncapsulation(Header, Payload) -> binary() when
37     Header :: binary(),
38     Payload :: binary().
39 pktEncapsulation(Header, Payload) ->
40     Ipv6Pckt = getIpv6Pkt(Header, Payload),
41     DhTypebinary = <<?IPV6_DHTYPE:8, 0:16>>,
42     <<DhTypebinary/binary, Ipv6Pckt/binary>>.
43
44 %%
45 %% @doc Encapsulates an Uncompressed IPv6 packet
46 %% @spec getUncIpv6(Ipv6Pckt) -> binary().
47
48 -spec getUncIpv6(Ipv6Pckt) -> binary() when
49     Ipv6Pckt :: binary().
50 getUncIpv6(Ipv6Pckt) ->
51     <<?IPV6_DHTYPE:8, Ipv6Pckt/bitstring>>.
```

```

52
53 -----
54 %
55 %                               Header compression
56 %
57 -----
58
59 -----
60 %% @doc Compresses an Ipv6 packet header according to the IPHC compression scheme
61 %% @spec compressIpv6Header(Ipv6Pckt, RouteExist) -> {binary(), map()}.
62 %% @returns a tuple containing the compressed header in binary form
63 %% and the values that should be carried inline
64 -----
65 -spec compressIpv6Header(Ipv6Pckt, RouteExist) -> {binary(), map()} when
66     Ipv6Pckt :: binary(),
67     RouteExist :: boolean().
68 compressIpv6Header(Ipv6Pckt, RouteExist) ->
69     PcktInfo = getPcktInfo(Ipv6Pckt),
70
71     TrafficClass = PcktInfo#ipv6PckInfo.trafficClass,
72     FlowLabel = PcktInfo#ipv6PckInfo.flowLabel,
73     NextHeader = PcktInfo#ipv6PckInfo.nextHeader,
74     HopLimit = PcktInfo#ipv6PckInfo.hopLimit,
75     SourceAddress = PcktInfo#ipv6PckInfo.sourceAddress,
76     DestAddress = PcktInfo#ipv6PckInfo.destAddress,
77
78     Map = #{},
79     List = [],
80
81     {CID, UpdateMap0, UpdatedList0} =
82         encodeCid(SourceAddress, DestAddress, Map, List),
83
84     {TF, UpdateMap1, UpdatedList1} =
85         encodeTf(TrafficClass, FlowLabel, UpdateMap0, UpdatedList0),
86
87     {NH, UpdateMap2, UpdatedList2} = encodeNh(NextHeader, UpdateMap1, UpdatedList1
88     ),
89
90     {HLIM, UpdateMap3, UpdatedList3} = encodeHlim(HopLimit, UpdateMap2,
91     UpdatedList2),
92
93     SAC = encodeSac(SourceAddress),
94
95     {SAM, UpdateMap4, UpdatedList4} =
96         encodeSam(CID, SAC, SourceAddress, UpdateMap3, UpdatedList3, RouteExist),
97
98     M = encodeM(DestAddress),
99
100     DAC = encodeDac(DestAddress),
101
102     {DAM, CarrInlineMap, CarrInlineList} =
103         encodeDam(CID, M, DAC, DestAddress, UpdateMap4, UpdatedList4, RouteExist),
104
105     %CH = {TF, NH, HLIM, CID, SAC, SAM, M, DAC, DAM, CarrInlineList},
106
107     CarrInlineBin = list_to_binary(CarrInlineList),
108     % io:format("Actual carried values: ~p ~n",[CarrInlineMap]),
109     case NextHeader of
110     ?UDP_PN ->
111         UdpPckt = getUdpData(Ipv6Pckt),
112         CompressedUdpHeaderBin = compressUdpHeader(UdpPckt, []),
113         io:format("Lowpan core: UdpInline: ~p ~n",[CompressedUdpHeaderBin]),

```

```

112         CompressedHeader =
113             <<?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>>,
114         {CompressedHeader, CarrInlineMap};
115     - ->
116     CompressedHeader =
117         <<?IPHC_DHTYPE:3, TF:2, NH:1, HLIM:2, CID:1, SAC:1, SAM:2, M:1,
                DAC:1, DAM:2, CarrInlineBin/binary>>,
118         {CompressedHeader, CarrInlineMap}
119     end.
120
121 %-----
122 %% @private
123 %% @doc Encodes the TrafficClass and Flow label fields
124 %% @spec encodeTf(TrafficClass, FlowLabel, CarrInlineMap, CarrInlineList) -> {
        integer(), map(), list()}.
125 %% @returns a tuple containing the compressed values and the CarrInline values
126 %-----
127 -spec encodeTf(TrafficClass, FlowLabel, CarrInlineMap, CarrInlineList) -> {term(),
        map(), list()} when
128     TrafficClass :: integer(),
129     FlowLabel :: integer(),
130     CarrInlineMap :: map(),
131     CarrInlineList :: list().
132 encodeTf(TrafficClass, FlowLabel, CarrInlineMap, CarrInlineList) ->
133     <<DSCP:6, ECN:2>> = <<TrafficClass:8>>,
134
135     case {ECN, DSCP, FlowLabel} of
136     {0, 0, 0} ->
137         % Traffic Class and Flow Label are elided
138         {2#11, CarrInlineMap, CarrInlineList};
139
140     {_, 0, _} ->
141         % DSCP is elided
142         UpdatedMap = CarrInlineMap#{"ECN" => ECN, "FlowLabel" => FlowLabel},
143         Bin = <<ECN:2, 0:2, FlowLabel:20>>, % 24 bits tot (RFC 6282 - pg12)
144         L = [Bin],
145         UpdatedList = [CarrInlineList, L],
146         {2#01, UpdatedMap, UpdatedList};
147
148     {_, _, 0} ->
149         % Flow Label is elided
150         UpdatedMap = CarrInlineMap#{"TrafficClass" => TrafficClass},
151         Bin = <<ECN:2, DSCP:6>>,
152         L = [Bin],
153         UpdatedList = [CarrInlineList, L],
154         {2#10, UpdatedMap, UpdatedList};
155
156     - ->
157         % ECN, DSCP, and Flow Label are carried inline
158         UpdatedMap = CarrInlineMap#{"TrafficClass" => TrafficClass, "FlowLabel"
                => FlowLabel},
159         Bin = <<ECN:2, DSCP:6, 0:4, FlowLabel:20>>, % 32 bits tot (RFC 6282 -
                pg12)
160         L = [Bin],
161         UpdatedList = [CarrInlineList, L],
162         {2#00, UpdatedMap, UpdatedList}
163     end.
164
165 %-----
166 %% @private

```

```

167 %% @doc Encodes the NextHeader field
168 %% @doc NextHeader specifies whether or not the next header is encoded using NHC
169 %% @spec encodeNh(NextHeader, CarrInlineMap, CarrInlineList)->{integer(), map(),
    list()}.
170 %% @returns a tuple containing the compressed value and the CarrInline values
171 %-----
172 -spec encodeNh(NextHeader, CarrInlineMap, CarrInlineList) -> {integer(), map(),
    list()} when
173     NextHeader :: integer(),
174     CarrInlineMap :: map(),
175     CarrInlineList :: list().
176 encodeNh(NextHeader, CarrInlineMap, CarrInlineList) when NextHeader == ?UDP_PN ->
177     {1, CarrInlineMap, CarrInlineList};
178 encodeNh(NextHeader, CarrInlineMap, CarrInlineList) when NextHeader == ?TCP_PN ->
179     Bin = <<NextHeader:8>>,
180     L = [Bin],
181     UpdatedList = [CarrInlineList, L],
182     {0, CarrInlineMap#{"NextHeader" => ?TCP_PN}, UpdatedList};
183 encodeNh(NextHeader, CarrInlineMap, CarrInlineList) when NextHeader == ?ICMP_PN ->
184     Bin = <<NextHeader:8>>,
185     L = [Bin],
186     UpdatedList = [CarrInlineList, L],
187     {0, CarrInlineMap#{"NextHeader" => ?ICMP_PN}, UpdatedList};
188 encodeNh(NextHeader, CarrInlineMap, CarrInlineList) ->
189     Bin = <<NextHeader:8>>,
190     L = [Bin],
191     UpdatedList = [CarrInlineList, L],
192     {0, CarrInlineMap#{"NextHeader" => NextHeader}, UpdatedList}.
193
194 %-----
195 %% @private
196 %% @doc Encodes the HopLimit field
197 %% @spec encodeHlim(HopLimit, CarrInlineMap, CarrInlineList) -> {integer(), map(),
    list()}.
198 %% @returns a tuple containing the compressed value and the CarrInline values
199 %-----
200 -spec encodeHlim(HopLimit, CarrInlineMap, CarrInlineList) -> {integer(), map(),
    list()} when
201     HopLimit :: integer(),
202     CarrInlineMap :: map(),
203     CarrInlineList :: list().
204 encodeHlim(HopLimit, CarrInlineMap, CarrInlineList) when HopLimit == 1 ->
205     {2#01, CarrInlineMap, CarrInlineList};
206 encodeHlim(HopLimit, CarrInlineMap, CarrInlineList) when HopLimit == 64 ->
207     {2#10, CarrInlineMap, CarrInlineList};
208 encodeHlim(HopLimit, CarrInlineMap, CarrInlineList) when HopLimit == 255 ->
209     {2#11, CarrInlineMap, CarrInlineList};
210 encodeHlim(HopLimit, CarrInlineMap, CarrInlineList) ->
211     Bin = <<HopLimit:8>>,
212     L = [Bin],
213     UpdatedList = CarrInlineList ++ L,
214     {2#00, CarrInlineMap#{"HopLimit" => HopLimit}, UpdatedList}.
215
216 %-----
217 %% @private
218 %% @doc Encodes the Context Identifier Extension field
219 %% @doc If this bit is 1, an 8 bit CIE field follows after the DAM field
220 %% @spec encodeCid(SrcAdd, DstAdd, CarrInlineMap, CarrInlineList) -> {integer(),
    map(), list()}.
221 %% @returns a tuple containing the compressed value and the CarrInline values
222 %-----

```

```

223 -spec encodeCid(SrcAdd, DstAdd, CarrInlineMap, CarrInlineList) -> {integer(), map
    (), list()} when
224     SrcAdd :: binary(),
225     DstAdd :: binary(),
226     CarrInlineMap :: map(),
227     CarrInlineList :: list().
228 encodeCid(SrcAdd, DstAdd, CarrInlineMap, CarrInlineList) ->
229     <<SrcAddPrefix:16, _/binary>> = <<SrcAdd:128>>,
230     <<DstAddPrefix:16, _/binary>> = <<DstAdd:128>>,
231     SrcPrefixKey = <<SrcAddPrefix:16, 0:48>>,
232     DstPrefixKey = <<DstAddPrefix:16, 0:48>>,
233
234     % check if prefix is in contextTable
235     SrcContext = maps:find(SrcPrefixKey, ?PrefixId_table),
236     DstContext = maps:find(DstPrefixKey, ?PrefixId_table),
237
238     case {SrcContext, DstContext} of
239     {{ok, SrcContextId}, {ok, DstContextId}} ->
240         Bin = <<SrcContextId:4, DstContextId:4>>,
241         L = [Bin],
242         UpdatedList = CarrInlineList ++ L,
243         {1, CarrInlineMap, UpdatedList};
244
245     {error, {ok, DstContextId}} ->
246         Bin = <<0:4, DstContextId:4>>,
247         L = [Bin],
248         UpdatedList = CarrInlineList ++ L,
249         {1, CarrInlineMap, UpdatedList};
250
251     {{ok, SrcContextId}, error} ->
252         SrcContextId = someValue,
253         Bin = <<SrcContextId:4, 0:4>>,
254         L = [Bin],
255         UpdatedList = CarrInlineList ++ L,
256         {1, CarrInlineMap, UpdatedList};
257
258     _ -> {0, CarrInlineMap, CarrInlineList}
259     end.
260
261 %-----
262 %% @private
263 %% @doc Encodes the Source Address Compression
264 %% @doc SAC specifies whether the compression is stateless or statefull
265 %% @spec encodeSac(SrcAdd) -> integer().
266 %% @returns the compressed value
267 %-----
268 -spec encodeSac(SrcAdd) -> integer() when
269     SrcAdd :: binary().
270 encodeSac(SrcAdd) ->
271     <<Prefix:16, _/binary>> = <<SrcAdd:128>>,
272
273     case Prefix of
274     ?LINK_LOCAL_PREFIX ->
275         0;
276     ?MULTICAST_PREFIX ->
277         0;
278     _ ->
279         1
280     end.
281
282 %-----
283 %% @private

```



```

284 %% @doc Encodes for the Source Address Mode
285 %% @spec encodeSam(integer(), integer(), binary(), map(), list(), boolean()) -> {
integer(), map(), list()}.
286 %% @returns a tuple containing the compressed value and the CarrInline values
287 %-----
288 -spec encodeSam(integer(), integer(), binary(), map(), list(), boolean()) -> {
integer(), map(), list()}.
289 encodeSam(_CID, SAC, SrcAdd, CarrInlineMap, CarrInlineList, RouteExist) when SAC
== 0 ->
290   SrcAddBits = <<SrcAdd:128>>,
291   <<_:112, Last16Bits:16>> = SrcAddBits,
292   <<_:64, Last64Bits:64>> = SrcAddBits,
293
294   case {SrcAddBits, RouteExist} of
295   <<?LINK_LOCAL_PREFIX:16, 0:48, _:24, 16#FFFE:16, _:24>>, _} ->
296     % the address is fully elided
297     {2#11, CarrInlineMap, CarrInlineList};
298   {_, true} ->
299     {2#11, CarrInlineMap, CarrInlineList};
300
301   <<?LINK_LOCAL_PREFIX:16, 0:48, 16#000000FFFE00:48, _:16>>, _} ->
302     % the first 112 bits are elided, last 16 IID bits are carried in-line
303     Bin = <<Last16Bits:16>>,
304     L = [Bin],
305     UpdatedList = [CarrInlineList, L],
306     UpdatedMap = CarrInlineMap#{"SAM" => Last16Bits},
307     {2#10, UpdatedMap, UpdatedList};
308
309   <<?LINK_LOCAL_PREFIX:16, 0:48, _:64>>, _} ->
310     % the first 64 bits are elided, last 64 bits (IID) are carried in-line
311     Bin = <<Last64Bits:64>>,
312     L = [Bin],
313     UpdatedList = [CarrInlineList, L],
314     UpdatedMap = CarrInlineMap#{"SAM" => Bin},
315     {2#01, UpdatedMap, UpdatedList};
316   {_, _} ->
317     % full address is carried in-line
318     Bin = <<SrcAdd:128>>,
319     L = [Bin],
320     UpdatedList = [CarrInlineList, L],
321     {2#00, CarrInlineMap#{"SAM" => Bin}, UpdatedList}
322   end;
323 encodeSam(0, 1, SrcAdd, CarrInlineMap, CarrInlineList, _RouteExist) ->
324   Bin = <<SrcAdd:128>>,
325   L = [Bin],
326   UpdatedList = [CarrInlineList, L],
327   {2#00, CarrInlineMap#{"SAM" => Bin}, UpdatedList};
328
329 encodeSam(_CID, SAC, SrcAdd, CarrInlineMap, CarrInlineList, _RouteExist) when SAC
== 1 ->
330   SrcAddBits = <<SrcAdd:128>>,
331   <<_:112, Last16Bits:16>> = SrcAddBits,
332   <<_:64, Last64Bits:64>> = SrcAddBits,
333
334   case SrcAddBits of
335   <<_Prefix:16, _:48, _:24, 16#FFFE:16, _:24>> ->
336     % the address is fully elided
337     {2#11, CarrInlineMap, CarrInlineList};
338
339   <<_Prefix:16, _:48, 16#000000FFFE00:48, _:16>> ->
340     % the first 112 bits are elided, last 16 IID bits are carried in-line
341     Bin = <<Last16Bits:16>>,

```

```

342         L = [Bin],
343         UpdatedList = [CarrInlineList, L],
344         UpdatedMap = CarrInlineMap#{"SAM" => Bin},
345         {2#10, UpdatedMap, UpdatedList};
346
347     <<_Prefix:16, _:48, _:64>> ->
348         % the first 64 bits are elided, last 64 bits (IID) are carried in-line
349         Bin = <<Last64Bits:64>>,
350         L = [Bin],
351         UpdatedList = [CarrInlineList, L],
352         UpdatedMap = CarrInlineMap#{"SAM" => Bin},
353         {2#01, UpdatedMap, UpdatedList};
354
355     <<0:128>> -> % The UNSPECIFIED address, ::
356         {2#00, CarrInlineMap, CarrInlineList}
357 end.
358
359 %-----
360 %% @private
361 %% @doc Defines the multicast compression
362 %% @spec encodeM(DstAdd) -> integer().
363 %% @returns the compressed value
364 %-----
365 -spec encodeM(DstAdd) -> integer() when
366     DstAdd :: binary().
367 encodeM(DstAdd) ->
368     <<Prefix:16, _/bitstring>> = <<DstAdd:128>>,
369     case Prefix of
370         ?MULTICAST_PREFIX ->
371             1;
372         _ ->
373             0
374     end.
375
376 %-----
377 %% @private
378 %% @doc encode for the Destination Address Compression
379 %% @spec encodeDac(DstAdd) -> integer().
380 %% @doc DAC specifies whether the compression is stateless or statefull
381 %% @returns the compressed value
382 %-----
383 -spec encodeDac(DstAdd) -> integer() when
384     DstAdd :: binary().
385 encodeDac(DstAdd) ->
386     <<Prefix:16, _/binary>> = <<DstAdd:128>>,
387
388     case Prefix of
389         ?LINK_LOCAL_PREFIX ->
390             0;
391         ?MULTICAST_PREFIX ->
392             0;
393         _ ->
394             1
395     end.
396
397 %-----
398 %% @private
399 %% @doc Encodes logic for the Destination Address Mode
400 %% @spec encodeDam(integer(), integer(), integer(), binary(), map(),
401 %% list(), boolean()) -> {integer(), map(), list()}.
402 %% @param Cid, M, DAC, DstAdd, CarrInlineMap
403 %% @returns a tuple containing the compressed value and the CarrInline values

```

```

404 %-----
405 -spec encodeDam(integer(), integer(), integer(), binary(), map(), list(), boolean
    ()) -> {integer(), map(), list()}.
406 encodeDam(0, 0, 0, DstAdd, CarrInlineMap, CarrInlineList, RouteExist) ->
407   DestAddBits = <<DstAdd:128>>,
408   <<_:112, Last16Bits:16>> = DestAddBits,
409   <<_:64, Last64Bits:64>> = DestAddBits,
410
411   case {DestAddBits, RouteExist} of
412     {<<?LINK_LOCAL_PREFIX:16, 0:48, _:24, 16#FFFE:16, _:24>>, _} ->
413       % MAC address is split into two 24-bit parts, FFFE is inserted in the
         middle
414       {2#11, CarrInlineMap, CarrInlineList};
415     {_, true} -> {2#11, CarrInlineMap, CarrInlineList};
416
417     {<<?LINK_LOCAL_PREFIX:16, 0:48, 16#000000FFFE00:48, _:16>>, _} ->
418       % the first 112 bits are elided, last 16 bits are in-line
419       Bin = <<Last16Bits:16>>,
420       L = [Bin],
421       UpdatedList = [CarrInlineList, L],
422       UpdatedMap = CarrInlineMap#{"DAM" => Bin},
423       {2#10, UpdatedMap, UpdatedList};
424
425     {<<?LINK_LOCAL_PREFIX:16, 0:48, _:64>>, _} ->
426       % the first 64 bits are elided, last 64 bits are in-line
427       Bin = <<Last64Bits:64>>,
428       L = [Bin],
429       UpdatedList = [CarrInlineList, L],
430       UpdatedMap = CarrInlineMap#{"DAM" => Bin},
431       {2#01, UpdatedMap, UpdatedList};
432     {_, _} ->
433       % full address is carried in-line
434       Bin = <<DstAdd:128>>,
435       L = [Bin],
436       UpdatedList = [CarrInlineList, L],
437       {2#00, CarrInlineMap#{"DAM" => Bin}, UpdatedList}
438   end;
439 encodeDam(1, 0, 1, DstAdd, CarrInlineMap, CarrInlineList, _RouteExist) ->
440   DestAddBits = <<DstAdd:128>>,
441   <<_:112, Last16Bits:16>> = DestAddBits,
442   <<_:64, Last64Bits:64>> = DestAddBits,
443
444   case DestAddBits of
445     <<_Prefix:16, _:48, _:24, 16#FFFE:16, _:24>> ->
446       % the address is fully elided
447       {2#11, CarrInlineMap, CarrInlineList};
448
449     <<_Prefix:16, _:48, 16#000000FFFE00:48, _:16>> ->
450       % the first 112 bits are elided, last 16 IID bits are carried in-line
451       Bin = <<Last16Bits:16>>,
452       L = [Bin],
453       UpdatedList = [CarrInlineList, L],
454       UpdatedMap = CarrInlineMap#{"DAM" => Bin},
455       {2#10, UpdatedMap, UpdatedList};
456
457     <<_Prefix:16, _:48, _:64>> ->
458       % the first 64 bits are elided, last 64 bits (IID) are carried in-line
459       Bin = <<Last64Bits:64>>,
460       L = [Bin],
461       UpdatedList = [CarrInlineList, L],
462       UpdatedMap = CarrInlineMap#{"DAM" => Bin},
463       {2#01, UpdatedMap, UpdatedList}

```

```

464     end;
465 encodeDam(0, 0, 1, DstAdd, CarrInlineMap, CarrInlineList, _RouteExist) ->
466     Bin = <<DstAdd:128>>,
467     L = [Bin],
468     UpdatedList = [CarrInlineList, L],
469     {2#00, CarrInlineMap#{"DAM" => Bin}, UpdatedList};
470
471 encodeDam(_CID, 1, 0, DstAdd, CarrInlineMap, CarrInlineList, _RouteExist) ->
472     DestAddBits = <<DstAdd:128>>,
473     <<_:80, Last48Bits:48>> = DestAddBits,
474     <<_:96, Last32Bits:32>> = DestAddBits,
475     <<_:120, Last8Bits:8>> = DestAddBits,
476     case DestAddBits of
477     % ff02::00XX.
478     <<?MULTICAST_PREFIX:16, 0:104, _:8>> ->
479         Bin = <<Last8Bits:8>>,
480         L = [Bin],
481         UpdatedList = [CarrInlineList, L],
482         UpdatedMap = CarrInlineMap#{"DAM" => Bin},
483         {2#11, UpdatedMap, UpdatedList};
484
485     % ffXX::00XX:XXXX.
486     <<16#FF:8, _:8, 0:80, _:32>> ->
487         Bin = <<Last32Bits:32>>,
488         L = [Bin],
489         UpdatedList = [CarrInlineList, L],
490         UpdatedMap = CarrInlineMap#{"DAM" => Bin},
491         {2#10, UpdatedMap, UpdatedList};
492
493     % ffXX::00XX:XXXX:XXXX.
494     <<16#FF:8, _:8, 0:64, _:48>> ->
495         Bin = <<Last48Bits:48>>,
496         L = [Bin],
497         UpdatedList = [CarrInlineList, L],
498         UpdatedMap = CarrInlineMap#{"DAM" => Bin},
499         {2#01, UpdatedMap, UpdatedList};
500     - ->
501     % full address is carried in-line
502     Bin = <<DstAdd:128>>,
503     L = [Bin],
504     UpdatedList = [CarrInlineList, L],
505     {2#00, CarrInlineMap#{"DAM" => Bin}, UpdatedList}
506     end;
507 encodeDam(_CID, 1, 1, DstAdd, CarrInlineMap, CarrInlineList, _RouteExist) ->
508     DestAddBits = <<DstAdd:128>>,
509     <<_:80, Last48Bits:48>> = DestAddBits,
510     case DestAddBits of
511     <<16#FF, _:112>> ->
512         Bin = <<Last48Bits:48>>,
513         L = [Bin],
514         UpdatedList = [CarrInlineList, L],
515         UpdatedMap = CarrInlineMap#{"DAM" => Bin},
516         {2#00, UpdatedMap, UpdatedList}
517     end.
518
519 %-----
520 %
521 %           Next Header compression
522 %
523 %-----
524
525 %-----

```

```

526 %                               UDP Packet Compression
527 %-----
528
529 %-----
530 %                               Structure of a UDP Datagram Header
531 %
532 %           0                   1                   2                   3
533 %           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
534 % +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
535 % |           Source Port           |           Destination Port           |
536 % +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
537 % |           Length           |           Checksum           |
538 % +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
539 %
540
541 -spec compressUdpHeader(UdpPckt, CarriedInline) -> binary() when
542     UdpPckt :: binary(),
543     CarriedInline :: list().
544 compressUdpHeader(UdpPckt, CarriedInline) ->
545     <<SrcPort:16, DstPort:16, _Length:16, Checksum:16>> = <<UdpPckt:64>>,
546
547     {P, CarriedInlineList} = encodeUdpPorts(SrcPort, DstPort, CarriedInline),
548     {C, CarriedIn} = encodeUdpChecksum(Checksum, CarriedInlineList),
549
550     Inline = list_to_binary(CarriedIn),
551
552     CompressedUdpHeader = <<?UDP_DHTYPE:5, C:1, P:2, Inline/bitstring>>,
553     CompressedUdpHeader.
554
555 -spec encodeUdpPorts(SrcPort, DstPort, CarriedInline) -> {integer(), list()} when
556     SrcPort :: integer(),
557     DstPort :: integer(),
558     CarriedInline :: list().
559 encodeUdpPorts(SrcPort, DstPort, CarriedInline) ->
560     case {<<SrcPort:16>>, <<DstPort:16>>} of
561     {<<?0xf0b:12, Last4S_Bits:4>>, <<?0xf0b:12, Last4D_Bits:4>>} ->
562         ToCarr = <<Last4S_Bits:4, Last4D_Bits:4>>,
563         L = [ToCarr],
564         CarriedInlineList = CarriedInline ++ L,
565         P = 2#11,
566         {P, CarriedInlineList};
567     {<<?0xf0:8, Last8S_Bits:8>>, _} ->
568         ToCarr = <<Last8S_Bits:8, DstPort:16>>,
569         L = [ToCarr],
570         CarriedInlineList = CarriedInline ++ L,
571         P = 2#10,
572         {P, CarriedInlineList};
573     {_, <<?0xf0:8, Last8D_Bits:8>>} ->
574         ToCarr = <<SrcPort:16, Last8D_Bits:8>>,
575         L = [ToCarr],
576         CarriedInlineList = CarriedInline ++ L,
577         P = 2#01,
578         {P, CarriedInlineList};
579     {_, _} ->
580         P = 2#00,
581         ToCarr = <<SrcPort:16, DstPort:16>>,
582         L = [ToCarr],
583         CarriedInlineList = CarriedInline ++ L,
584         {P, CarriedInlineList}
585     end.
586
587 -spec encodeUdpChecksum(Checksum, CarriedInline) -> {integer(), list()} when

```

```

588     Checksum :: integer(),
589     CarriedInline :: list().
590 encodeUdpChecksum(Checksum, CarriedInline) ->
591     case Checksum of
592     0 ->
593         {1, CarriedInline};
594     %%Checksum is carried inline
595     _ ->
596         L = [<<Checksum:16>>],
597         UpdatedList = CarriedInline ++ L,
598         {0, UpdatedList}
599     end.
600
601 %-----
602 %                               ICMP Packet Compression
603 %-----
604
605 %-----
606 %                               TCP Packet Compression
607 %-----
608
609 %-----
610 %                               Packet Compression Helper
611 %-----
612
613 %-----
614 %% @doc Creates a compressed 6lowpan packet (with iphc compression) from an Ipv6
615 %% packet
616 %% @spec createIphcPckt(IphcHeader, Payload) -> binary().
617 -spec createIphcPckt(IphcHeader, Payload) -> binary() when
618     IphcHeader :: binary(),
619     Payload :: binary().
620 createIphcPckt(IphcHeader, Payload) ->
621     <<IphcHeader/binary, Payload/bitstring>>.
622
623 %-----
624 %% @doc Returns value field of a given Ipv6 packet
625 %% @spec getPcktInfo(Ipv6Pckt) -> map().
626 %-----
627 -spec getPcktInfo(Ipv6Pckt) -> map() when
628     Ipv6Pckt :: binary().
629 getPcktInfo(Ipv6Pckt) ->
630     <<Version:4, TrafficClass:8, FlowLabel:20, PayloadLength:16, NextHeader:8,
631         HopLimit:8, SourceAddress:128, DestAddress:128, Data/bitstring>> =
632         Ipv6Pckt,
633     Payload = case NextHeader of
634         ?UDP_PN ->
635             <<_UdpFields:64, Payld/bitstring>> = Data,
636             Payld;
637         _ -> Data
638     end,
639     PckInfo =
640     #ipv6PckInfo{
641         version = Version,
642         trafficClass = TrafficClass,
643         flowLabel = FlowLabel,
644         payloadLength = PayloadLength,
645         nextHeader = NextHeader,
646         hopLimit = HopLimit,
647         sourceAddress = SourceAddress,

```

```

648         destAddress = DestAddress,
649         payload = Payload
650     },
651     PckInfo.
652
653 %-----
654 %% @doc Returns value field of a decoded Ipv6 packet
655 %% @spec getDecodeIpv6PcktInfo(Ipv6Pckt) -> map().
656 %-----
657 -spec getDecodeIpv6PcktInfo(Ipv6Pckt) -> map() when
658     Ipv6Pckt :: binary().
659 getDecodeIpv6PcktInfo(Ipv6Pckt) ->
660     <<TrafficClass:8, FlowLabel:24, NextHeader:8, HopLimit:8, SourceAddress:128,
661         DestAddress:128, Data/bitstring>> =
662         Ipv6Pckt,
663
664     Payload = case NextHeader of
665         ?UDP_PN ->
666             <<_UdpFields:64, Payld/bitstring>> = Data,
667             Payld;
668         _ -> Data
669     end,
670     PckInfo =
671         #ipv6PcktInfo{
672             version = 6,
673             trafficClass = TrafficClass,
674             flowLabel = FlowLabel,
675             payloadLength = byte_size(Payload),
676             nextHeader = NextHeader,
677             hopLimit = HopLimit,
678             sourceAddress = SourceAddress,
679             destAddress = DestAddress,
680             payload = Payload
681         },
682     PckInfo.
683 %-----
684 %% @doc Returns UDP data from a given Ipv6 packet if it contains a UDP nextHeader
685 %% @spec getUdpData(Ipv6Pckt) -> binary().
686 %-----
687 -spec getUdpData(Ipv6Pckt) -> binary() when
688     Ipv6Pckt :: binary().
689 getUdpData(Ipv6Pckt) ->
690     <<_:320, UdpPckt:64, _/binary>> = Ipv6Pckt,
691     UdpPckt.
692
693 %-----
694 %% @doc Returns the payload of a given Ipv6 packet
695 %% @spec getIpv6Payload(Ipv6Pckt) -> binary().
696 %-----
697 -spec getIpv6Payload(Ipv6Pckt) -> binary() when
698     Ipv6Pckt :: binary().
699 getIpv6Payload(Ipv6Pckt) ->
700     <<_:192, _:128, Payload/bitstring>> = Ipv6Pckt,
701     Payload.
702
703 %-----
704 %% @doc Encodes an Integer value in a binary format using an appropriate amount of
705 %% bit
706 %% @spec encodeInteger(I) -> binary().
707 %-----
708 -spec encodeInteger(I) -> binary() when

```

```

708     I :: integer().
709 encodeInteger(I) when I =< 255 ->
710     <<I:8>>;
711 encodeInteger(I) when I =< 65535 ->
712     <<I:16>>;
713 encodeInteger(I) when I =< 4294967295 ->
714     <<I:32>>;
715 encodeInteger(I) ->
716     <<I:64>>.
717
718 %-----
719 %
720 %                               Packet fragmentation
721 %
722 %-----
723
724 %-----
725 %% @doc Builds subsequent fragment header
726 %% @spec buildFragHeader(FragHeader) -> binary().
727 %-----
728 -spec buildFragHeader(FragHeader) -> binary() when
729     FragHeader :: map().
730 buildFragHeader(FragHeader) ->
731     #frag_header{
732         frag_type = FragType,
733         datagram_size = DatagramSize,
734         datagram_tag = DatagramTag,
735         datagram_offset = DatagramOffset
736     } = FragHeader,
737     <<FragType:5, DatagramSize:11, DatagramTag:16, DatagramOffset:8>>.
738
739 %-----
740 %% @doc Builds first fragment header
741 %% @spec buildFirstFragHeader(FragHeader) -> binary().
742 %-----
743 -spec buildFirstFragHeader(FragHeader) -> binary() when
744     FragHeader :: map().
745 buildFirstFragHeader(FragHeader) ->
746     #frag_header{
747         frag_type = FragType,
748         datagram_size = DatagramSize,
749         datagram_tag = DatagramTag
750     } = FragHeader,
751     <<FragType:5, DatagramSize:11, DatagramTag:16>>.
752
753 %-----
754 %% @spec buildFirstFragPckt(FragType, DatagramSize, DatagramTag, CompressedHeader,
755     Payload) -> binary().
756 %-----
757 -spec buildFirstFragPckt(integer(), integer(), integer(), binary(), binary()) ->
758     binary().
759 buildFirstFragPckt(FragType, DatagramSize, DatagramTag, CompressedHeader, Payload)
760     ->
761     <<FragType:5, DatagramSize:11, DatagramTag:16, CompressedHeader/binary,
762     Payload/bitstring>>.
763
764 %-----
765 %% @doc Creates a fragmented packet
766 %% @spec buildDatagramPckt(DtgmHeader, Payload) -> binary().
767 %-----
768 -spec buildDatagramPckt(map(), binary()) -> binary().
769 buildDatagramPckt(DtgmHeader, Payload) ->

```



```

766     TYPE = DtgMHeader#frag_header.frag_type,
767     case TYPE of
768         ?FRAG1_DHTYPE ->
769             Header = buildFirstFragHeader(DtgMHeader),
770             <<Header/binary, Payload/bitstring>>;
771         ?FRAGN_DHTYPE ->
772             Header = buildFragHeader(DtgMHeader),
773             <<Header/binary, Payload/bitstring>>
774     end.
775
776 %-----
777 %% @doc Checks if a packet needs to be fragmented or not and has a valid size
778 %% returns a list of fragments if yes, the original packet if not
779 %% @spec triggerFragmentation(binary(), integer()) -> {boolean(), list()} | {atom()
780 %% @doc Checks if a packet needs to be fragmented or not and has a valid size
781 %% returns a list of fragments if yes, the original packet if not
782 %% @spec triggerFragmentation(binary(), integer()) -> {boolean(), list()}
783 %% | {size_err, error_frag_size}.
784 triggerFragmentation(CompPckt, DatagramTag, RouteExist) when byte_size(CompPckt)
785 =< ?MAX_DTG_SIZE ->
786     PcktLengt = byte_size(CompPckt),
787
788     ValidLength = PcktLengt =< ?MAX_FRAME_SIZE,
789     case ValidLength of
790         false ->
791             io:format("The received Ipv6 packet needs fragmentation to be
792             transmitted~n"),
793             Fragments = fragmentIpv6Packet(CompPckt, DatagramTag, RouteExist),
794             {true, Fragments};
795         true ->
796             io:format("No fragmentation needed~n"),
797             {false, CompPckt}
798     end;
799
800 triggerFragmentation(_CompPckt, _DatagramTag, _RouteExist) ->
801     {size_err, error_frag_size}.
802
803 %-----
804 %% @doc Fragments a given Ipv6 packet
805 %% @spec fragmentIpv6Packet(binary(), integer()) -> list().
806 %% @returns a list of fragmented packets having this form:
807 %% [{FragHeader1, Fragment1}, ..., {FragHeaderN, FragmentN}]
808 %-----
809 -spec fragmentIpv6Packet(binary(), integer(), boolean()) -> list().
810 fragmentIpv6Packet(CompIpv6Pckt, DatagramTag, RouteExist) when is_binary(
811     CompIpv6Pckt) ->
812     Size = byte_size(CompIpv6Pckt),
813     fragProcess(CompIpv6Pckt, DatagramTag, Size, 0, [], RouteExist).
814
815 %-----
816 %% @private
817 %% @doc helper function to process the received packet
818 %% @returns a list of fragmented packets
819 %% [{Header1, Fragment1}, ..., {HeaderN, FragmentN}]
820 %% @spec fragProcess(binary(), integer(), integer(), integer(), list()) -> list().
821 %% Input :
822 %%     Ipv6Pckt := binary
823 %%     Pckt size := integer
824 %%     DatagramTag := integer
825 %%     Offset := integer
826 %%     Accumulator : list
827 %-----

```

```

823 -spec fragProcess(binary(), integer(), integer(), integer(), list(), boolean()) ->
      list().
824 fragProcess(<<>>, _DatagramTag, _PacketLen, _Offset, Acc, _RouteExist) ->
825   lists:reverse(Acc);
826 fragProcess(CompIpv6Pckt, DatagramTag, PacketLen, Offset, Acc, RouteExist) ->
827   MaxSize = case RouteExist of
828     true-> ?MAX_FRAG_SIZE_MESH;
829     false -> ?MAX_FRAG_SIZE_NoMESH
830   end,
831   PcktSize = byte_size(CompIpv6Pckt),
832   FragmentSize = min(PcktSize, MaxSize),
833
834   <<FragPayload:FragmentSize/binary, Rest/bitstring>> = CompIpv6Pckt,
835
836   case Offset of
837     0 ->
838       Header =
839         buildFirstFragHeader(#frag_header{
840           frag_type = ?FRAGI_DHTYPE,
841           datagram_size = PacketLen,
842           datagram_tag = DatagramTag,
843           datagram_offset = Offset
844         });
845     _ ->
846       Header =
847         buildFragHeader(#frag_header{
848           frag_type = ?FRAGN_DHTYPE,
849           datagram_size = PacketLen,
850           datagram_tag = DatagramTag,
851           datagram_offset = Offset
852         })
853   end,
854
855   fragProcess(Rest, DatagramTag, PacketLen, Offset + 1, [{Header, FragPayload} |
      Acc], RouteExist).
856
857 %-----
858 %% @doc Check if tag exist in the map, if so generate a new one and update the tag
      map
859 %% @spec checkTagUnicity(map(), integer()) -> {integer(), map()}.
860 %-----
861 -spec checkTagUnicity(map(), integer()) -> {integer(), map()}.
862 checkTagUnicity(Map, Tag) ->
863   Exist = maps:is_key(Tag, Map),
864   case Exist of
865     true ->
866       NewTag = rand:uniform(?MAX_TAG_VALUE),
867       checkTagUnicity(Map, NewTag);
868     false ->
869       NewMap = maps:put(Tag, valid, Map),
870       {Tag, NewMap}
871   end.
872
873 %-----
874 %
875 %                               Packet Decoding
876 %
877 %-----
878
879 %-----
880 %% @doc decode an Ipv6 packet header compressed according to the IPHC compression
      scheme

```

```

881 %% @spec decodeIpv6Pckt(boolean(), binary(), binary(), binary()) -> binary() | {
      atom(), atom()}.
882 %% @returns the decoded Ipv6 packet
883 %-----
884 -spec decodeIpv6Pckt(boolean(), binary(), binary(), binary()) -> binary() | {atom
      (), atom()}.
885 decodeIpv6Pckt(RouteExist, OriginatorMacAddr, CurrNodeMacAddr, CompressedPacket) ->
886   <<Dispatch:3, TF:2, NH:1, HLIM:2, CID:1, SAC:1, SAM:2, M:1, DAC:1, DAM:2, Rest
      /bitstring>> =
887     CompressedPacket,
888     case Dispatch of
889     ?IPHC_DHTYPE ->
890       {SrcContextId, DstContextId, Rest0} = decodeCid(CID, Rest),
891       {{DSCP, ECN}, FlowLabel, Rest1} = decodeTf(TF, Rest0),
892       {NextHeader, Rest2} = decodeNextHeader(NH, Rest1),
893       {HopLimit, Rest3} = decodeHlim(HLIM, Rest2),
894       {SourceAddress, Rest4} = decodeSam(SAC, SAM, Rest3, OriginatorMacAddr,
      SrcContextId, RouteExist),
895       {DestAddress, Payload} = decodeDam(M, DAC, DAM, Rest4, CurrNodeMacAddr,
      DstContextId, RouteExist),
896       PayloadLength = byte_size(Payload),
897       TrafficClass = DSCP bsl 2 + ECN,
898
899       <<Header:5, Inline/bitstring>> = Payload,
900
901       io:format("-----~n"),
902       io:format("Decoded packet~n"),
903       io:format("-----~n"),
904       DecodedPckt =
905       case Header of
906       ?UDP_DHTYPE->
907         {SrcPort, DstPort, Checksum, UdpPayload} = decodeUdpPckt(
      Inline),
908         Length = byte_size(UdpPayload),
909         io:format("IPv6~n"),
910
911         io:format("Traffic class: ~p~nFlow label: ~p~nNext header: ~p~n
      Hop limit: ~p~nSource address: ~p~nDestination address: ~
      p~n",
912                 [TrafficClass, FlowLabel, NextHeader, HopLimit,
      convert(SourceAddress), convert(DestAddress)])
913
914         io:format("
      -----~n"),
915         io:format("UDP~n"),
916         io:format("Source port: ~p~nDestination Port: ~p~nLength: ~p~n
      Checksum: ~p~n",[ SrcPort, DstPort, Length, Checksum]),
917         io:format("
      -----~n"),
918         io:format("Data: ~p~n",[UdpPayload]),
919         io:format("
      -----~n"),
920         <<6:4, TrafficClass, FlowLabel:20, PayloadLength:16, NextHeader:8,
      HopLimit:8,
921         SourceAddress/binary, DestAddress/binary, SrcPort:16, DstPort
      :16, Length:16, Checksum:16, Payload/bitstring>>;
922
923     _->
924       io:format("IPv6~n"),
925       io:format("Traffic class: ~p~nFlow label: ~p~nPayload length:
      ~p~nNext header: ~p~nHop limit: ~p~nSource address: ~p~n

```

```

        nDestination address: ~p~nData: ~p~n", [TrafficClass,
        FlowLabel, PayloadLength,
926         NextHeader, HopLimit, convert(SourceAddress),
        convert(DestAddress), Payload]),
927         io:format("
        -----n"),
928         <<6:4,TrafficClass,FlowLabel:20,PayloadLength:16,NextHeader:8,
        HopLimit:8,
929         SourceAddress/binary, DestAddress/binary, Payload/bitstring>>
930     end,
931     DecodedPckt;
932
933     _-> error_decoding
934 end.
935
936 -----
937 %% @private
938 %% @doc Decode logic for the CID field
939 %% @spec decodeCid(integer(), binary()) -> {integer(), integer(), binary()}.
940 %% @returns the decoded ContextID
941 -----
942 -spec decodeCid(integer(), binary()) -> {integer(), integer(), binary()}.
943 decodeCid(CID, CarriedInline) when CID == 1 ->
944     <<SrcContextId:4, DstContextId:4, Rest/bitstring>>= CarriedInline,
945     {SrcContextId, DstContextId, Rest};
946 decodeCid(CID, CarriedInline) when CID == 0 ->
947     DefaultPrefix = 0,
948     {DefaultPrefix, DefaultPrefix, CarriedInline}.
949
950 -----
951 %% @private
952 %% @doc decode logic for the TF field
953 %% @spec decodeTf(integer(), binary()) -> {{integer(), integer()}, integer(),
954     binary()}.
955 %% @returns the decoded TrafficClass and FlowLabel value
956 -----
957 -spec decodeTf(integer(), binary()) -> {{integer(), integer()}, integer(), binary
958     ()}.
959 decodeTf(TF, CarriedInline) ->
960     case TF of
961         2#11 ->
962             ECN = 0, DSCP = 0, FL = 0,
963             {{DSCP, ECN}, FL, CarriedInline};
964         2#01 ->
965             <<ECN:2, _rsv:2, FL:20, Rest/bitstring>> = CarriedInline,
966             DSCP = 0,
967             {{DSCP, ECN}, FL, Rest};
968         2#10 ->
969             <<ECN:2, DSCP:6, Rest/bitstring>> = CarriedInline,
970             FL = 0,
971             {{DSCP, ECN}, FL, Rest};
972         2#00 ->
973             <<ECN:2, DSCP:6, _rsv:4, FL:20, Rest/bitstring>> = CarriedInline,
974             {{DSCP, ECN}, FL, Rest}
975     end.
976
977 -----
978 %% @private
979 %% @doc Decode logic for the NH field
980 %% @spec decodeNextHeader(integer(), binary()) -> {integer(), binary()}.
981 %% @returns the decoded NextHeader value

```

```

981 %-----
982 -spec decodeNextHeader(integer(), binary()) -> {integer(), binary()}.
983 decodeNextHeader(NH, CarriedInline) when NH == 0 ->
984     <<NextHeader:8, Rest/bitstring>> = CarriedInline,
985     {NextHeader, Rest};
986 decodeNextHeader(NH, CarriedInline) when NH == 1 ->
987     {?UDP_PN, CarriedInline}.
988
989 %-----
990 %% @private
991 %% @doc Decode logic for the HLim field
992 %% @spec decodeHlim(integer(), binary()) -> {integer(), binary()}.
993 %% @returns the decoded Hop Limit value
994 %-----
995 -spec decodeHlim(integer(), binary()) -> {integer(), binary()}.
996 decodeHlim(HLim, CarriedInline) ->
997     <<HopLimit:8, Rest/bitstring>> = CarriedInline,
998     case HLim of
999         2#11 ->
1000             {255, CarriedInline};
1001         2#10 ->
1002             {64, CarriedInline};
1003         2#01 ->
1004             {1, CarriedInline};
1005         2#00 ->
1006             {HopLimit, Rest}
1007     end.
1008
1009 %-----
1010 %% @private
1011 %% @doc decode logic for the SAC field
1012 %% @spec decodeSam(integer(), integer(), binary(), binary(), integer(), boolean())
1013 %% @returns the decoded Source Address Mode value
1014 %-----
1015 -spec decodeSam(integer(), integer(), binary(), binary(), integer(), boolean()) ->
1016     {binary(), binary()}.
1017 decodeSam(SAC, SAM, CarriedInline, MacIID, _Context, RouteExist) when SAC == 0 ->
1018     case {SAM, RouteExist} of
1019         {2#11, true} ->
1020             SrcAdd = <<?LINK_LOCAL_PREFIX:16, 0:48, MacIID/binary>>,
1021             {SrcAdd, CarriedInline};
1022         {2#11, false} ->
1023             <<_:48, IID:16>> = MacIID,
1024             SrcAdd = <<?LINK_LOCAL_PREFIX:16, 0:48, 0:16, 16#00FF:16, 16#FE00:16,
1025                 IID:16>>,
1026             {SrcAdd, CarriedInline};
1027         {2#10, _} ->
1028             <<Last16Bits:16, Rest/bitstring>> = CarriedInline,
1029             SrcAdd = <<?LINK_LOCAL_PREFIX:16, 0:48, 16#000000FFFE00:48, Last16Bits
1030                 :16>>,
1031             {SrcAdd, Rest};
1032         {2#01, _} ->
1033             <<Last64Bits:64, Rest/bitstring>> = CarriedInline,
1034             SrcAdd = <<?LINK_LOCAL_PREFIX:16, 0:48, Last64Bits:64>>,
1035             {SrcAdd, Rest};
1036         {2#00, _} ->
1037             <<SrcAdd:128, Rest/bitstring>> = CarriedInline,
1038             {SrcAdd, Rest}
1039     end;
1040 decodeSam(SAC, _SAM, CarriedInline, _MacIID, 0, _RouteExist) when SAC == 1 ->
1041     <<SrcAdd:128, Rest/bitstring>> = CarriedInline,

```

```

1039     {<<SrcAdd:128>>, Rest};
1040 decodeSam(SAC, SAM, CarriedInline, MacIID, Context, _RouteExist) when SAC == 1 ->
1041     SrcAddrPrefix = maps:get(Context, ?Context_id_table),
1042     case SAM of
1043     2#11 ->
1044         <<_:48, IID:16>> = MacIID,
1045         SrcAdd = <<SrcAddrPrefix/binary, 0:16, 16#00FF:16, 16#FE00:16, IID
1046             :16>>,
1047         {SrcAdd, CarriedInline};
1048     2#10 ->
1049         <<Last16Bits:16, Rest/bitstring>> = CarriedInline,
1050         SrcAdd = <<SrcAddrPrefix/binary, 16#000000FFFE00:48, Last16Bits:16>>,
1051         {SrcAdd, Rest};
1052     2#01 ->
1053         <<Last64Bits:64, Rest/bitstring>> = CarriedInline,
1054         SrcAdd = <<SrcAddrPrefix/binary, Last64Bits:64>>,
1055         {SrcAdd, Rest};
1056     2#00 ->
1057         SrcAdd = <<0:128>>,
1058         {SrcAdd, CarriedInline}
1059     end.
1060 %-----
1061 %% @private
1062 %% @doc Decode logic for the DAC field
1063 %% @spec decodeDam(integer(), integer(), integer(), binary(), binary(), integer(),
1064     boolean()) -> {binary(), binary()}.
1065 %% @returns the decoded Destination Address Mode value
1066 %-----
1067 -spec decodeDam(integer(), integer(), integer(), binary(), binary(), integer(),
1068     boolean()) -> {binary(), binary()}.
1069 decodeDam(0, 0, DAM, CarriedInline, MacIID, _Context, RouteExist) ->
1070     case {DAM, RouteExist} of
1071     {2#11, true} ->
1072         DstAdd = <<?LINK_LOCAL_PREFIX:16, 0:48, MacIID/binary>>,
1073         {DstAdd, CarriedInline};
1074     {2#11, false} ->
1075         DstAdd = <<?LINK_LOCAL_PREFIX:16, 0:48, 0:24, 16#FFFE:16, 0:24>>,
1076         {DstAdd, CarriedInline};
1077     {2#10, _} ->
1078         <<Last16Bits:16, Rest/bitstring>> = CarriedInline,
1079         DstAdd = <<?LINK_LOCAL_PREFIX:16, 0:48, 16#000000FFFE00:48, Last16Bits
1080             :16>>,
1081         {DstAdd, Rest};
1082     {2#01, _} ->
1083         <<Last64Bits:64, Rest/bitstring>> = CarriedInline,
1084         DstAdd = <<?LINK_LOCAL_PREFIX:16, 0:48, Last64Bits:64>>,
1085         {DstAdd, Rest};
1086     {2#00, _} ->
1087         <<DstAdd:128, Rest/bitstring>> = CarriedInline,
1088         {DstAdd, Rest}
1089     end;
1090 decodeDam(0, 1, _DAM, CarriedInline, _MacIID, 0, _RouteExist) ->
1091     <<DstAdd:128, Rest/bitstring>> = CarriedInline,
1092     {<<DstAdd:128>>, Rest};
1093 decodeDam(0, 1, DAM, CarriedInline, _MacIID, Context, _RouteExist) ->
1094     DstAddrPrefix = maps:get(Context, ?Context_id_table),
1095     case DAM of
1096     2#11 ->
1097         {<<DstAddrPrefix/binary, 0:24, 16#FFFE:16, 0:24>>, CarriedInline};
1098     2#10 ->
1099         <<Last16Bits:16, Rest/bitstring>> = CarriedInline,

```

```

1097     DstAdd = <<DstAddrPrefix/binary, 16#000000FFFE00:48, Last16Bits:16>>,
1098     {DstAdd, Rest};
1099     2#01 ->
1100     <<Last64Bits:64, Rest/bitstring>> = CarriedInline,
1101     DstAdd = <<DstAddrPrefix/binary, Last64Bits:64>>,
1102     {DstAdd, Rest};
1103     2#00 -> {error_reserved, CarriedInline}
1104 end;
1105 decodeDam(1, 0, DAM, CarriedInline, _MacIID, _Context, _RouteExist) ->
1106 case DAM of
1107     2#11 ->
1108     <<Last8Bits:8, Rest/bitstring>> = CarriedInline,
1109     DstAdd = <<?MULTICAST_PREFIX:16, 0:104, Last8Bits>>,
1110     {DstAdd, Rest};
1111     2#10 ->
1112     <<Last32Bits:32, Rest/bitstring>> = CarriedInline,
1113     DstAdd = <<?MULTICAST_PREFIX:16, 0:80, Last32Bits:32>>,
1114     {DstAdd, Rest};
1115     2#01 ->
1116     <<Last48Bits:48, Rest/bitstring>> = CarriedInline,
1117     DstAdd = <<?MULTICAST_PREFIX:16, 0:64, Last48Bits:48>>,
1118     {DstAdd, Rest};
1119     2#00 ->
1120     <<DstAdd:128, Rest/bitstring>> = CarriedInline,
1121     {DstAdd, Rest}
1122 end;
1123 decodeDam(1, 1, DAM, CarriedInline, _MacIID, _Context, _RouteExist) ->
1124 case DAM of
1125     2#00 ->
1126     <<Last48Bits:48, Rest/bitstring>> = CarriedInline,
1127     DstAdd = <<16#FF:16, 0:64, Last48Bits:48>>,
1128     {DstAdd, Rest}
1129 end.
1130
1131 -spec decodeUdpPckt(binary()) -> {integer(), integer(), integer(), binary()}.
1132 decodeUdpPckt(Rest) ->
1133 <<C:1, P:2, Inline/bitstring>> = Rest,
1134 {SrcPort, DstPort, Rest1} = decodePort(P, Inline),
1135 {Checksum, Payload} = decodeChecksum(C, Rest1),
1136 {SrcPort, DstPort, Checksum, Payload}.
1137
1138 -spec decodePort(integer(), binary()) -> {integer(), integer(), binary()}.
1139 decodePort(P, Inline) ->
1140 case P of
1141     2#11 ->
1142     <<Last4S_Bits:4, Last4D_Bits:4, Rest/bitstring>> = Inline,
1143     SrcPort = <<?0xf0b:12, Last4S_Bits:4>>,
1144     DstPort = <<?0xf0b:12, Last4D_Bits:4>>,
1145     <<S:16>> = SrcPort,
1146     <<D:16>> = DstPort,
1147     {S, D, Rest};
1148     2#10 ->
1149     <<Last8S_Bits:8, DstPort:16, Rest/bitstring>> = Inline,
1150     SrcPort = <<?0xf0:8, Last8S_Bits:8>>,
1151     <<S:16>> = SrcPort,
1152     {S, DstPort, Rest};
1153     2#01 ->
1154     <<SrcPort:16, Last8D_Bits:8, Rest/bitstring>> = Inline,
1155     DstPort = <<?0xf0:8, Last8D_Bits:8>>,
1156     <<D:16>> = DstPort,
1157     {SrcPort, D, Rest};
1158     2#00 ->

```

```

1159         <<SrcPort:16, DstPort:16, Rest/bitstring>> = Inline,
1160         {SrcPort, DstPort, Rest}
1161     end.
1162
1163 -spec decodeChecksum(integer(), binary()) -> {integer(), binary()}.
1164 decodeChecksum(C, Inline) ->
1165     case C of
1166     1 -> {0, Inline};
1167     0 ->
1168         <<Checksum:16, Rest/bitstring>> = Inline,
1169         {Checksum, Rest}
1170     end.
1171
1172 %-----
1173 %                               Packet Decompression Helper
1174 %-----
1175
1176 -spec convertAddrToBin(term()) -> binary().
1177 convertAddrToBin(Address) ->
1178     DestAdd = case is_integer(Address) of
1179     true ->
1180         encodeInteger(Address);
1181     false ->
1182         Address
1183     end,
1184     DestAdd.
1185
1186 -spec tupleToBin(tuple()) -> binary().
1187 tupleToBin(Tuple) ->
1188     Elements = tuple_to_list(Tuple),
1189     Binaries = [elementToBinary(Elem) || Elem <- Elements],
1190     list_to_binary(Binaries).
1191
1192 -spec elementToBinary(term()) -> binary().
1193 elementToBinary(Elem) when is_integer(Elem) ->
1194     encodeInteger(Elem);
1195 elementToBinary(Elem) when is_binary(Elem) ->
1196     Elem;
1197 elementToBinary(Elem) when is_tuple(Elem) ->
1198     tupleToBin(Elem);
1199 elementToBinary(Elem) when is_list(Elem) ->
1200     list_to_binary(Elem).
1201
1202 %-----
1203 %
1204 %                               Reassembly
1205 %
1206 %-----
1207
1208 %-----
1209 %% @spec datagramInfo(binary()) -> map().
1210 %% @doc helper function to retrieve datagram info
1211 %% @returns a tuple containing useful fragment info
1212 %-----
1213 -spec datagramInfo(binary()) -> map().
1214 datagramInfo(Fragment) ->
1215     <<FragType:5, Rest/bitstring>> = Fragment,
1216     case FragType of
1217     ?FRAG1_DHTYPE ->
1218         <<DatagramSize:11, DatagramTag:16, Payload/bitstring>> = Rest,
1219         FragInfo =
1220             #datagramInfo{

```



```

1221         fragtype = FragType,
1222         datagramSize = DatagramSize,
1223         datagramTag = DatagramTag,
1224         datagramOffset = 0,
1225         payload = Payload
1226     },
1227     FragInfo;
1228 ?FRAGN_DHTYPE ->
1229     <<DatagramSize:11, DatagramTag:16, DatagramOffset:8, Payload/bitstring
1230     >> = Rest,
1231     FragInfo =
1232         #datagramInfo{
1233             fragtype = FragType,
1234             datagramSize = DatagramSize,
1235             datagramTag = DatagramTag,
1236             datagramOffset = DatagramOffset,
1237             payload = Payload
1238         },
1239     FragInfo
1240 end.
1241
1242
1243 %-----
1244 %% @doc Stores fragment in ETS and check if the datagram is complete
1245 %% @spec storeFragment(atom(), term(), integer(), binary(), integer(), integer(),
1246 %% integer(), term()) -> {term(), map()}.
1247 %-----
1248 -spec storeFragment(map(), term(), integer(), binary(), integer(), integer(),
1249 integer(), term()) -> {term(), map()}.
1250 storeFragment(DatagramMap, Key, Offset, Payload, CurrTime, Size, Tag, _From) ->
1251 {Result, Map} = case ets:lookup(DatagramMap, Key) of
1252 [] ->
1253     handleNewDatagram(DatagramMap, Key, Offset, Payload, CurrTime, Size,
1254 Tag);
1255 [{Key, OldDatagram}] ->
1256     handleExistingDatagram(DatagramMap, Key, Offset, Payload, CurrTime,
1257 Size, OldDatagram)
1258 end,
1259
1260 io:format("-----~n"),
1261 io:format("DatagramMap after update:~n"),
1262 printDatagramMap(DatagramMap),
1263 io:format("-----~n"),
1264 {Result, Map}.
1265
1266 -spec handleNewDatagram(map(), term(), integer(), binary(), integer(), integer(),
1267 integer()) -> {term(), map()}.
1268 handleNewDatagram(DatagramMap, Key, Offset, Payload, CurrTime, Size, Tag) ->
1269 if byte_size(Payload) == Size ->
1270     ReassembledPacket = reassemble(#datagram{
1271 tag = Tag,
1272 size = Size,
1273 cmpt = byte_size(Payload),
1274 fragments = #{Offset => Payload},
1275 timer = CurrTime
1276 },),
1277     ets:insert(DatagramMap, {Key, ReassembledPacket}),
1278     {complete_first_frag, ReassembledPacket};
1279 true ->
1280     NewDatagram = #datagram{
1281 tag = Tag,

```

```

1277         size = Size,
1278         cmpt = byte_size(Payload),
1279         fragments = #{Offset => Payload},
1280         timer = CurrTime
1281     },
1282     ets:insert(DatagramMap, {Key, NewDatagram}),
1283     {incomplete_first, Key}
1284 end.
1285
1286 -spec handleExistingDatagram(map(), term(), integer(), binary(), integer(),
1287 integer(), map()) -> {term(), map()}.
1287 handleExistingDatagram(DatagramMap, Key, Offset, Payload, CurrTime, Size,
1288 OldDatagram) ->
1288     Fragments = OldDatagram#datagram.fragments,
1289     case maps:is_key(Offset, Fragments) of
1290     true ->
1291         {duplicate, OldDatagram};
1292     false ->
1293         NewFragments = maps:put(Offset, Payload, Fragments),
1294         NewCmpt = OldDatagram#datagram.cmpt + byte_size(Payload),
1295         UpdatedDatagram = OldDatagram#datagram{
1296             cmpt = NewCmpt,
1297             fragments = NewFragments,
1298             timer = CurrTime
1299         },
1300         ets:insert(DatagramMap, {Key, UpdatedDatagram}),
1301         if NewCmpt == Size ->
1302             {complete, UpdatedDatagram};
1303         true ->
1304             {incomplete, UpdatedDatagram}
1305         end
1306     end.
1307
1308 -spec printDatagramMap(map()) -> ok.
1309 printDatagramMap(DatagramMap) ->
1310     List = ets:tab2list(DatagramMap),
1311     lists:foreach(fun({Key, Value}) -> printEntry(Key, Value) end, List).
1312
1313 -spec printEntry(term(), tuple()) -> ok.
1314 printEntry(Key, {datagram, Tag, Size, Cmpt, Timer, Fragments}) ->
1315     io:format("~p -> {datagram, ~p, ~p, ~p, ~n    #{~n", [Key, Tag, Size, Cmpt]),
1316     printFragments(Fragments),
1317     io:format("    }, ~p}~n", [Timer]).
1318
1319 -spec printFragments(map()) -> ok.
1320 printFragments(Fragments) ->
1321     maps:fold(fun(Offset, Payload, Acc) ->
1322         io:format("        ~p => ~p,~n", [Offset, Payload]),
1323         Acc
1324     end, ok, Fragments).
1325
1326 %-----
1327 %% @spec reassemble(map()) -> binary().
1328 %% @doc Reassemble the datagram from stored fragments
1329 %-----
1330 -spec reassemble(map()) -> binary().
1331 reassemble(Datagram) ->
1332     FragmentsMap = Datagram#datagram.fragments,
1333     SortedFragments =
1334         lists:sort([{Offset, Fragment} || {Offset, Fragment} <- maps:to_list(
1335             FragmentsMap)]),
1335     lists:foldl(

```

```

1336     fun({_Offset, Payload}, Acc) ->
1337         <<Acc/binary, Payload/binary>>
1338     end,
1339     <<>>,
1340     SortedFragments
1341 ).
1342
1343 %-----
1344 %
1345 %                               ROUTING
1346 %
1347 %-----
1348
1349 %-----
1350 %                               Mesh Addressing Type and Header
1351 %
1352 %           0                   1                   2                   3
1353 %           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
1354 %           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
1355 %           |1 0|V|F|HopsLft|  originator address,final destination address
1356 %           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
1357 %
1358
1359 %-----
1360 %% @doc Creates mesh header binary
1361 %% @spec buildMeshHeader(map()) -> binary().
1362 %-----
1363 -spec buildMeshHeader(map()) -> binary().
1364 buildMeshHeader(MeshHeader) ->
1365     #mesh_header{
1366         v_bit = VBit,
1367         f_bit = FBit,
1368         hops_left = HopsLeft,
1369         originator_address = OriginatorAddress,
1370         final_destination_address = FinalDestinationAddress
1371     } = MeshHeader,
1372     <<?MESH_DHTYPE:2, VBit:1, FBit:1, HopsLeft:4,
1373         OriginatorAddress/binary, FinalDestinationAddress/binary>>.
1374
1375 %-----
1376 %% @spec createNewMeshDatagram(binary(), binary(), binary()) -> binary().
1377 %% @doc Creates new mesh header and returns new datagram
1378 %-----
1379 -spec createNewMeshDatagram(binary(), binary(), binary()) -> binary().
1380 createNewMeshDatagram(Datagram, SenderMacAdd, DstMacAdd) ->
1381     VBit =
1382         if
1383             byte_size(SenderMacAdd) == 8 -> 0;
1384             true -> 1
1385         end,
1386     FBit =
1387         if
1388             byte_size(DstMacAdd) == 8 -> 0;
1389             true -> 1
1390         end,
1391
1392     MeshHeader =
1393         #mesh_header{
1394             v_bit = VBit,
1395             f_bit = FBit,
1396             hops_left = ?Max_Hops,
1397             originator_address = SenderMacAdd,

```

```

1398         final_destination_address = DstMacAdd
1399     },
1400     BinMeshHeader = buildMeshHeader(MeshHeader),
1401     <<BinMeshHeader/binary, Datagram/bitstring>>.
1402
1403 %-----
1404 %% @doc Creates new mesh header
1405 %% @spec createNewMeshHeader(binary(), binary(), boolean()) -> binary().
1406 %-----
1407 createNewMeshHeader(SenderMacAdd, DstMacAdd, Extended_hopsleft) ->
1408     VBit =
1409         if
1410             byte_size(SenderMacAdd) == 8 -> 0;
1411             true -> 1
1412         end,
1413     FBit =
1414         if
1415             byte_size(DstMacAdd) == 8 -> 0;
1416             true -> 1
1417         end,
1418
1419     case Extended_hopsleft of
1420         true ->
1421             <<?MESH_DHTYPE:2, VBit:1, FBit:1, ?DeepHopsLeft:4,
1422             SenderMacAdd/binary, DstMacAdd/binary, ?Max_DeepHopsLeft:8>>;
1423         false ->
1424             <<?MESH_DHTYPE:2, VBit:1, FBit:1, ?Max_Hops:4,
1425             SenderMacAdd/binary, DstMacAdd/binary>>
1426     end.
1427
1428 %-----
1429 %% @doc Returns routing info in mesh header
1430 %% @spec getMeshInfo(binary()) -> map().
1431 %-----
1432 -spec getMeshInfo(binary()) -> map().
1433 getMeshInfo(Datagram) ->
1434     <<_:2, _V:1, _F:1, Hops_left:4, _/bitstring>> = Datagram,
1435
1436     case Hops_left of
1437         ?DeepHopsLeft ->
1438             <<?MESH_DHTYPE:2, VBit:1, FBit:1, HopsLeft:4, OriginatorAddress:64,
1439             FinalDestinationAddress:64, DeepHopsLeft:8, Data/bitstring>> =
1440             Datagram;
1441         - ->
1442             <<?MESH_DHTYPE:2, VBit:1, FBit:1, HopsLeft:4, OriginatorAddress:64,
1443             FinalDestinationAddress:64, Data/bitstring>> =
1444             Datagram,
1445             DeepHopsLeft = undefined
1446     end,
1447     MeshInfo =
1448         #meshInfo{
1449             v_bit = VBit,
1450             f_bit = FBit,
1451             hops_left = HopsLeft,
1452             originator_address = <<OriginatorAddress:64>>,
1453             final_destination_address = <<FinalDestinationAddress:64>>,
1454             deep_hops_left = DeepHopsLeft,
1455             payload = Data
1456         },
1457     MeshInfo.
1458 %-----

```

```

1458 %% @doc Checks if datagram in mesh type, if so return true and mesh header info
1459 %% @spec containsMeshHeader(binary()) -> {boolean(), map()} | boolean().
1460 %-----
1461 -spec containsMeshHeader(binary()) -> {boolean(), map()} | boolean().
1462 containsMeshHeader(Datagram) ->
1463     case Datagram of
1464         <<Dispatch:2, _/bitstring>> when Dispatch == ?MESH_DHTYPE ->
1465             {true, getMeshInfo(Datagram)};
1466         _ ->
1467             false
1468     end.
1469
1470 %-----
1471 %% @doc Removes mesh header if the datagram was meshed (used in put and
1472 %% reassemble)
1473 %% @spec removeMeshHeader(binary(), integer()) -> binary().
1474 %-----
1475 -spec removeMeshHeader(binary(), integer()) -> binary().
1476 removeMeshHeader(Datagram, HopsLeft) ->
1477     case Datagram of
1478         <<?MESH_DHTYPE:2, _/bitstring>> ->
1479             case HopsLeft of
1480                 ?DeepHopsLeft ->
1481                     <<?MESH_DHTYPE:2, _Header:142, Rest/bitstring>> = Datagram
1482                     ,
1483                     Rest;
1484                 _ ->
1485                     <<?MESH_DHTYPE:2, _Header:134, Rest/bitstring>> = Datagram
1486                     ,
1487                     Rest
1488             end;
1489         _ ->
1490             Datagram
1491     end.
1492
1493 %-----
1494 %% @doc Checks the next hop in the routing table and create new datagram with mesh
1495 %% header if meshing is needed
1496 %% @spec getNextHop(binary(), binary(), binary(), binary(), integer(),
1497 %% boolean()) -> {boolean(), binary(), map()} | {boolean(), binary(), map(), map()}
1498 %-----
1499 -spec getNextHop(CurrNodeMacAdd, SenderMacAdd, DestMacAddress, DestAddress, SeqNum,
1500     HopsleftExtended) ->
1501     {boolean(), binary(), mac_header()}
1502     when
1503         CurrNodeMacAdd :: binary(),
1504         SenderMacAdd :: binary(),
1505         DestMacAddress :: binary(),
1506         DestAddress :: binary(),
1507         SeqNum :: integer(),
1508         HopsleftExtended :: boolean().
1509 getNextHop(CurrNodeMacAdd, SenderMacAdd, DestMacAddress, DestAddress, SeqNum,
1510     Hopsleft_extended) ->
1511     case <<DestAddress:128>> of
1512         <<16#FF:8, _/binary>> ->
1513             MulticastAddr = generateMulticastAddr(<<DestAddress:128>>),
1514             Multicast_EU64 = generateEUI64MacAddr(MulticastAddr),
1515             MHdr = #mac_header{src_addr = CurrNodeMacAdd, dest_addr =
1516                 Multicast_EU64},
1517             BroadcastHeader = createBroadcastHeader(SeqNum),

```

```

1513     MeshHdrBin = createNewMeshHeader(SenderMacAdd, DestMacAddress,
1514     Hopsleft_extended),
1515     Header = <<MeshHdrBin/bitstring, BroadcastHeader/bitstring>>,
1516     {false, Header, MHdr};
1517   _->
1518     case routing_table:getRoute(DestMacAddress) of
1519     NextHopMacAddr when NextHopMacAddr /= DestMacAddress ->
1520       io:format("Next hop found: ~p~n", [NextHopMacAddr]),
1521       MacHdr = #mac_header{src_addr = CurrNodeMacAdd, dest_addr =
1522         NextHopMacAddr},
1523       MeshHdrBin = createNewMeshHeader(SenderMacAdd, DestMacAddress,
1524       Hopsleft_extended),
1525       {true, MeshHdrBin, MacHdr};
1526     NextHopMacAddr when NextHopMacAddr == DestMacAddress ->
1527       io:format("Direct link found ~n"),
1528       MHdr = #mac_header{src_addr = CurrNodeMacAdd, dest_addr =
1529         DestMacAddress},
1530       {false, <<>>, MHdr};
1531     _ ->
1532       {false, <<>>, undefined, undefined}
1533   end
1534 end.
1535
1536 -spec getNextHop(binary(), binary()) -> {boolean(), binary(), map()} | {boolean(),
1537   binary(), map(), map()}.
1538 getNextHop(CurrNodeMacAdd, DestMacAddress) ->
1539   case routing_table:getRoute(DestMacAddress) of
1540   NextHopMacAddr when NextHopMacAddr /= DestMacAddress ->
1541     MacHdr = #mac_header{src_addr = CurrNodeMacAdd, dest_addr = NextHopMacAddr
1542     },
1543     MeshHdrBin = createNewMeshHeader(CurrNodeMacAdd, DestMacAddress, ?
1544     DeepHopsLeft),
1545     {true, MeshHdrBin, MacHdr};
1546   NextHopMacAddr when NextHopMacAddr == DestMacAddress ->
1547     MHdr = #mac_header{src_addr = CurrNodeMacAdd, dest_addr = DestMacAddress},
1548     {false, <<>>, MHdr};
1549   _ ->
1550     {false, <<>>, undefined, undefined}
1551   end.
1552
1553 -spec generateEUI64MacAddr(binary()) -> binary().
1554 generateEUI64MacAddr(MacAddress) when byte_size(MacAddress) == ?SHORT_ADDR_LEN ->
1555   PanID = <<16#FFFF:16>>,
1556   Extended48Bit = <<PanID/binary, 0:16, MacAddress/binary>>,
1557   <<A:8, Rest:40>> = Extended48Bit,
1558   ULBSetup = A band 16#FD,
1559   <<First:16, Last:24>> = <<Rest:40>>,
1560   EUI64 = <<ULBSetup:8, First:16, 16#FF:8, 16#FE:8, Last:24>>,
1561   EUI64;
1562 generateEUI64MacAddr(MacAddress) when byte_size(MacAddress) == ?EXTENDED_ADDR_LEN
1563 ->
1564   <<A:8, Rest:56>> = MacAddress,
1565   NewA = A bxor 2,
1566   <<NewA:8, Rest:56>>.
1567
1568 -spec getEUI64From48bitMac(binary()) -> binary().
1569 getEUI64From48bitMac(MacAddress) ->
1570   <<First:24, Last:24>> = MacAddress,
1571   <<A:8, Rest:16>> = <<First:24>>,
1572   NewA = A bxor 2,
1573   EUI64 = <<NewA:8, Rest:16, 16#fffe:16, Last:24>>,
1574   EUI64.

```

```

1567
1568
1569 -spec generateLLAddr(binary()) -> binary().
1570 generateLLAddr(MacAddress) ->
1571     EUI64 = generateEUI64MacAddr(MacAddress),
1572     LLAddr = <<16#FE80:16, 0:48, EUI64/binary>>,
1573     LLAddr.
1574
1575 -spec getEUI64MacAddr(binary()) -> binary().
1576 getEUI64MacAddr(Address) ->
1577     <<_:64, MacAddr:64/bitstring>> = <<Address:128>>,
1578     MacAddr.
1579
1580 -spec get16bitMacAddr(binary()) -> binary().
1581 get16bitMacAddr(Address) ->
1582     <<_:112, MacAddr:16/bitstring>> = <<Address:128>>,
1583     MacAddr.
1584
1585
1586 %-----
1587 % Generates a EUI64 address from the 16bit short mac address
1588 %-----
1589 getEUI64FromShortMac(MacAddress)->
1590     PanID = <<16#FFFF:16>>,%ieee802154:get_pib_attribute(mac_pan_id),
1591     Extended48Bit = <<PanID/binary, 0:16, MacAddress/binary>>,
1592     <<A:8, Rest:40>> = Extended48Bit,
1593     ULBSetup = A band 16#FD, % replace 7th bit of first byte (U/L) by 0
1594     <<First:16, Last:24>> = <<Rest:40>>,
1595     EUI64 = <<ULBSetup:8, First:16, 16#FF:8, 16#FE:8, Last:24>>,
1596     EUI64.
1597
1598 %-----
1599 % Generates a EUI64 address from the 64bit extended mac address
1600 %-----
1601 getEUI64FromExtendedMac(MacAddress)->
1602     <<A:8, Rest:56>> = MacAddress,
1603     NewA = A bxor 2,
1604     <<NewA:8, Rest:56>>.
1605
1606 -spec generateMulticastAddr(binary()) -> binary().
1607 generateMulticastAddr(DestAddress) ->
1608     <<_:112, DST_15:8, DST_16:8>> = DestAddress,
1609     <<_:3, Last5Bits:5>> = <<DST_15:8>>,
1610     MulticastAddr = <<2#100:3, Last5Bits:5, DST_16:8>>,
1611     MulticastAddr.
1612
1613 -spec createBroadcastHeader(integer()) -> binary().
1614 createBroadcastHeader(SeqNum) ->
1615     BCO_Header = <<?BCO_DHTYPE, SeqNum:8>>,
1616     BCO_Header.
1617
1618
1619 %-----
1620 %
1621 %                               Utils functions
1622 %
1623 %-----
1624
1625 -spec convert(binary()) -> list().
1626 convert(Binary) ->
1627     lists:flatten(
1628         lists:join(":",

```

```

1629         [io_lib:format("~2.16.0B", [B]) || <<B:8>> <= Binary]
1630     )
1631 ).
1632
1633 -spec generateChunks() -> binary().
1634 generateChunks() ->
1635     NumChunks = 5,
1636     ChunkSize = 58,
1637     Chunks =
1638         lists:map(fun(N) -> generateChunk(N, ChunkSize) end, lists:seq(NumChunks,
1639             1, -1)),
1639     Result = lists:foldl(fun(A, B) -> <<A/binary, B/binary>> end, <<>>, Chunks),
1640     Result.
1641
1642 -spec generateChunks(integer()) -> binary().
1643 generateChunks(Size) ->
1644     ChunkSize = 48,
1645     Chunks =
1646         lists:map(fun(N) -> generateChunk(N, ChunkSize) end, lists:seq(Size, 1,
1647             -1)),
1647     Result = lists:foldl(fun(A, B) -> <<A/binary, B/binary>> end, <<>>, Chunks),
1648     Result.
1649
1650 -spec generateChunk(integer(), integer()) -> binary().
1651 generateChunk(N, Size) ->
1652     Prefix = list_to_binary(io_lib:format("chunk_~2..0B", [N])),
1653     PrefixSize = byte_size(Prefix),
1654     PaddingSize = Size - PrefixSize,
1655     Padding = list_to_binary(lists:duplicate(PaddingSize, $a)),
1656     <<Prefix/binary, Padding/binary>>.

```

A.5 Lowpan API code

```

1 -module(lowpan_api).
2
3 -behaviour(gen_statem).
4
5 -include("lowpan.hrl").
6
7 -export([init/1, start_link/1, start/1, stop_link/0, stop/0]).
8 -export([callback_mode/0]).
9 -export([sendPacket/1, sendPacket/2, sendUncDatagram/3, tx/3, extendedHopsleftTx/1
10 ]).
11 -export([frameReception/0]).
12 -export([inputCallback/4]).
13 -export([idle/3]).
14 -export([tx_frame/3]).
15 -export([tx_datagram/3]).
16 -export([tx_packet/3]).
17 -export([rx_frame/3]).
18 -export([collect/3]).
19 -export([reassemble/3]).
20 -export([forward/3]).
21 -export([tx_packet_metrics/3]).
22
23 % API Functions
24 %% @doc Initializes the lowpan API module.

```



```

24 %% @spec init(map()) -> {ok, atom(), map()}.
25 init(Params) ->
26   io:format("
-----
n"),
27   io:format("Initialization-n"),
28   MacAddr = maps:get(node_mac_addr, Params),
29   CurrNodeMacAddr = lowpan_core:generateEUI64MacAddr(MacAddr),
30   io:format("Current node address: ~p-n",[CurrNodeMacAddr]),
31   setup_node_info_ets(),
32
33   RoutingTable = maps:get(routing_table, Params),
34
35   case routing_table:start_link(RoutingTable) of
36     {ok, _Pid} ->
37       io:format("-p: Routing table server successfully launched-n", [node()])
38       ;
39     {error, Reason} ->
40       io:format("-p: Failed to start routing table server: ~p-n", [node(),
41       Reason]),
42       exit({error, Reason})
43   end,
44
45   ieee802154_setup(CurrNodeMacAddr),
46
47   DatagramMap = ets:new(datagram_map, [named_table, public]),
48
49   Data = #{node_mac_addr => CurrNodeMacAddr, datagram_map => DatagramMap,
50   fragment_tag => ?DEFAULT_TAG_VALUE, seqNum => ?BC_SEQNUM,
51   metrics => #metrics{}, ack_req => false},
52
53   set_nodeData_value(state_data, Data),
54
55   io:format("-p: 6lowpan layer successfully launched-n", [node()]),
56   io:format("
-----
n"),
57   {ok, idle, Data}.
58 %% @doc Starts the lowpan API process linked to the current process.
59 %% @spec start_link(map()) -> {ok, pid()} | {error, Reason}.
60 start_link(Params) ->
61   gen_statem:start_link({local, ?MODULE}, ?MODULE, Params, []).
62
63 %% @doc Starts the lowpan API process.
64 %% @spec start(map()) -> {ok, pid()} | {error, Reason}.
65 start(Params) ->
66   gen_statem:start({local, ?MODULE}, ?MODULE, Params, []).
67
68
69 %% @doc Stops the lowpan API process linked to the current process.
70 %% @spec stop_link() -> ok.
71 stop_link() ->
72   gen_statem:stop(?MODULE).
73
74 %% @doc Stops the lowpan API process.
75 %% @spec stop() -> ok.
76 stop() ->
77   io:format("lowpan layer stopped"),
78   erpc:call(node(), routing_table, stop, []),
79   gen_statem:stop(?MODULE).

```

```

80
81 -----
82 %% @doc API function to send an IPv6 packet.
83 %% @spec sendPacket(binary()) -> ok | {error_multicast_src} | {
      error_unspecified_addr}.
84 -----
85 sendPacket(Ipv6Pckt) ->
86     io:format("Transmission request~n"),
87     PcktInfo = lowpan_core:getPcktInfo(Ipv6Pckt),
88     SrcAddress = PcktInfo#ipv6PckInfo.sourceAddress,
89     DstAddress = PcktInfo#ipv6PckInfo.destAddress,
90
91     case {<<SrcAddress:128>>, <<DstAddress:128>>} of
92     {<<16#FF:16, _:112>>, _} ->
93         io:format("Error, Source address cannot be a multicast address~n"),
94         error_multicast_src;
95     {_, <<0:128>>} ->
96         io:format("Error, destination address cannot be the Unspecified
97             address~n"),
98             error_unspecified_addr;
99     _ ->
100         Extended_hopsleft = false,
101         gen_statem:cast(?MODULE, {pckt_tx, Ipv6Pckt, PcktInfo,
102             Extended_hopsleft, self()}),
103         receive
104             Response ->
105                 Response
106         end
107     end.
108 -----
109 %% @doc API function to send an IPv6 packet with performance metrics enabled.
110 %% @spec sendPacket(binary()) -> ok | {error_multicast_src} | {
      error_unspecified_addr}.
111 -----
112 sendPacket(Ipv6Pckt, MetricEnabled) ->
113     io:format("Transmission request~n"),
114     PcktInfo = lowpan_core:getPcktInfo(Ipv6Pckt),
115     SrcAddress = PcktInfo#ipv6PckInfo.sourceAddress,
116     DstAddress = PcktInfo#ipv6PckInfo.destAddress,
117
118     case {<<SrcAddress:128>>, <<DstAddress:128>>} of
119     {<<16#FF:16, _:112>>, _} ->
120         io:format("Error, Source address cannot be a multicast address~n"),
121         error_multicast_src;
122     {_, <<0:128>>} ->
123         io:format("Error, destination address cannot be the Unspecified
124             address~n"),
125             error_unspecified_addr;
126     _ ->
127         Extended_hopsleft = false,
128         Response = case MetricEnabled of
129             true ->
130                 gen_statem:cast(?MODULE, {pckt_tx_with_metrics,
131                     Ipv6Pckt, PcktInfo, Extended_hopsleft, self()}
132                 );
133             false ->
134                 gen_statem:cast(?MODULE, {pckt_tx, Ipv6Pckt,
135                     PcktInfo, Extended_hopsleft, self()})
136         end,
137         receive
138             Response ->
139                 Response;
140     end.

```

```

134         {ok, NewMetrics} ->
135             {ok, RTT, SuccessRate, CompressionRatio} = handle_ack(
136                 NewMetrics),
137             _MetricsResult = {RTT, SuccessRate, CompressionRatio},
138             io:format("-----Metrics report
139                 ~~~~~~\n"),
140             io:format("RTT: ~p ms~nSuccessRate: ~p~nCompressionRatio: ~p~n
141                 ", [RTT, SuccessRate, CompressionRatio]),
142             io:format("
143                 ~~~~~~\n"),
144             ok;
145         error_frag_size ->
146             error_frag_size
147     end
148 end.
149
150 %-----
151 %% @doc API function to send an IPv6 packet with extended hops left option enabled
152 .
153 %% @spec extendedHopsleftTx(binary()) -> ok | {error_multicast_src} | {
154     error_unspecified_addr} | {error_timeout}.
155 %-----
156 extendedHopsleftTx(Ipv6Pckt) ->
157     io:format("New packet transmission ~n"),
158     PcktInfo = lowpan_core:getPcktInfo(Ipv6Pckt),
159     SrcAddress = PcktInfo#ipv6PcktInfo.sourceAddress,
160     DstAddress = PcktInfo#ipv6PcktInfo.destAddress,
161
162     case {<<SrcAddress:128>>, <<DstAddress:128>>} of
163     {<<?MULTICAST_PREFIX:16, _Rest:112>>, _} ->
164         io:format("Error, Source address cannot be a multicast address~n"),
165         error_multicast_src;
166     {_, <<0:128>>} ->
167         io:format("Error, destination address cannot be the Unspecified
168             address~n"),
169         error_unspecified_addr;
170     - ->
171         Extended_hopsleft = true,
172         Response = gen_statem:cast(?MODULE, {pckt_tx, Ipv6Pckt, PcktInfo,
173             Extended_hopsleft, self()}),
174         receive
175             error_frag_size ->
176                 error_frag_size;
177             Response ->
178                 Response
179         end
180     end.
181
182 %-----
183 %% @doc API function to send an uncompressed IPv6 datagram.
184 %% @spec sendUncDatagram(binary(), term(), map()) -> ok | {error_timeout}.
185 %-----
186 sendUncDatagram(Ipv6Pckt, FrameControl, MacHeader) ->
187     gen_statem:cast(?MODULE, {datagram_tx, Ipv6Pckt, FrameControl, MacHeader, self
188         ()}),
189     receive
190         Response ->
191             Response
192     after 1000 ->
193         io:format("Timeout~n"),
194         error_timeout
195     end.

```

```

187
188 %-----
189 %% @doc API function to send a frame.
190 %% @spec tx(binary(), term(), map()) -> ok | error_nalp.
191 %-----
192 tx(Frame, FrameControl, MacHeader) ->
193   case Frame of
194     <<?NALP_DHTYPE, _/bitstring>> ->
195       io:format("The received frame is not a lowpan frame~n"),
196       error_nalp;
197     _->
198       gen_statem:cast(?MODULE, {frame_tx, Frame, FrameControl, MacHeader,
199         self()}),
200       receive
201         Response ->
202           Response
203       end
204   end.
205 %-----
206 %% @doc API function to handle frame reception
207 %% @spec frameReception() -> term().
208 %-----
209 frameReception() ->
210   io:format("Reception mode~n"),
211   gen_statem:cast(?MODULE, {frame_rx, self()}),
212   receive
213     {reassembled_packet, IsMeshedPckt, OriginatorMacAddr, CurrNodeMacAdd,
214       ReassembledPacket} ->
215       io:format("Datagram reassembled, start packet decoding ~n"),
216       _DecodedPacket = lowpan_core:decodeIpv6Pckt(IsMeshedPckt,
217         OriginatorMacAddr, CurrNodeMacAdd, ReassembledPacket),
218       ReassembledPacket;
219     dtg_discarded ->
220       io:format("Datagram successfully discarded ~n"),
221       dtg_discarded;
222     {reassembly_timeout, DatagramMap, EntryKey} ->
223       io:format("Reassembly timeout for entry ~p~n", [EntryKey]),
224       ets:delete(DatagramMap, EntryKey),
225       io:format("Entry deleted~n"),
226       reassembly_timeout;
227     error_nalp->
228       error_nalp
229   after ?REASSEMBLY_TIMEOUT ->
230     reassembly_timeout
231   end.
232 %-----
233 % Input callback to handle new received frame.
234 %-----
235 inputCallback(Frame, _, _, _) ->
236   {FC, MH, Datagram} = Frame,
237   {IsMeshedPckt, FinalDstMacAdd, MeshPckInfo} = case lowpan_core:
238     containsMeshHeader(Datagram) of
239       {true, MeshInfo} ->
240         {true, MeshInfo#meshInfo.final_destination_address, MeshInfo};
241       false ->
242         {false, MH#mac_header.dest_addr, #{}}
243   end,
244   OriginatorAddr = case MeshPckInfo of
245     #{}-> MH#mac_header.src_addr;

```

```

245         _ -> MeshPckInfo#meshInfo.originator_address
246         end,
247
248     StateData = get_nodeData_value(state_data),
249
250     processFrame(IsMeshedPckt, MeshPckInfo, OriginatorAddr, FinalDstMacAdd, FC, MH
251         , Datagram, StateData).
252 %-----
253 %% @doc Processes new frame.
254 %% @spec handleDatagram(boolean(), map(), binary(), binary(), term(), map(),
255     binary(), map()) -> term().
256 %-----
257 processFrame(IsMeshedPckt, MeshPckInfo, OriginatorAddr, FinalDstMacAdd, FC, MH,
258     Datagram, StateData) ->
259     DestAdd = lowpan_core:convertAddrToBin(FinalDstMacAdd),
260     #{node_mac_addr := CurrNodeMacAdd} = StateData,
261
262     case DestAdd of
263     CurrNodeMacAdd ->
264         io:format("New frame received~n"),
265         io:format("Originator           : ~p~n",[OriginatorAddr]),
266         io:format("Final destination address: ~p~n", [DestAdd]),
267         io:format("Current node address      : ~p~n", [CurrNodeMacAdd]),
268
269         io:format("Final destination node reached, Forwarding to lowpan layer~
270             n"),
271         case IsMeshedPckt of
272         true ->
273             HopsLeft = MeshPckInfo#meshInfo.hops_left,
274             Rest = lowpan_core:removeMeshHeader(Datagram,HopsLeft),
275             gen_statem:cast(?MODULE, {new_frame_rx, IsMeshedPckt,
276                 OriginatorAddr, Rest});
277         false->
278             HopsLeft = 1,
279             Rest = lowpan_core:removeMeshHeader(Datagram,HopsLeft),
280             gen_statem:cast(?MODULE, {new_frame_rx, IsMeshedPckt,
281                 OriginatorAddr, Rest})
282
283         end;
284     ?BroadcastAdd ->
285         {keep_state, rx_frame};
286     _ ->
287         io:format("New frame received~n"),
288         io:format("Originator           : ~p~n",[OriginatorAddr]),
289         io:format("Final destination address: ~p~n", [DestAdd]),
290         io:format("Current node address      : ~p~n", [CurrNodeMacAdd]),
291         io:format("The datagram needs to be meshed~n"),
292         gen_statem:cast(?MODULE, {forward, Datagram, IsMeshedPckt, MeshPckInfo
293             , FinalDstMacAdd, CurrNodeMacAdd, FC, MH})
294
295     end.
296
297 %----- States -----
298 %-----
299 %% @doc In this state the machine waits to received transmission/reception request
300 %% @spec idle(atom(), term(), map()) -> {next_state, atom(), map(), list()}.
301 %-----
302 idle(cast, {pckt_tx, Ipv6Pckt, PcktInfo, Extended_hopsleft, From}, Data) ->
303     {next_state, tx_packet, Data#{data => {Ipv6Pckt, PcktInfo, Extended_hopsleft,
304         From}}, [{next_event, internal, {tx_packet}}]};
305

```

```

299 idle(cast, {pkt_tx_with_metrics, Ipv6Pckt, PcktInfo, Extended_hopsleft, From},
      Data) ->
300   {next_state, tx_packet_metrics, Data#{data => {Ipv6Pckt, PcktInfo,
      Extended_hopsleft, From}}, [{next_event, internal, {tx_packet_metrics}}]};
301
302 idle(cast, {frame_tx, Frame, FrameControl, MacHeader, From}, Data) ->
303   {next_state, tx_frame, Data#{data => {Frame, FrameControl, MacHeader, From}},
      [{next_event, internal, {tx_frame}}]};
304
305
306 idle(cast, {datagram_tx, Ipv6Pckt, FrameControl, MacHeader, From}, Data) ->
307   {next_state, tx_datagram, Data#{data => {Ipv6Pckt, FrameControl, MacHeader,
      From}}, [{next_event, internal, {tx_datagram}}]};
308
309 idle(cast, {frame_rx, From}, Data) ->
310   {next_state, rx_frame, Data#{caller => From}, [{next_event, internal, {
      rx_frame}}]}.
311
312
313 %----- Tx frame state -----
314
315 %-----
316 %% @doc Handles the transmission of a frame.
317 %% @spec tx_frame(atom(), term(), map()) -> {next_state, atom(), map()}.
318 %-----
319 tx_frame(internal, {tx_frame}, Data) ->
320   #{data := {Frame, FrameControl, MacHeader, From}} = Data,
321   Transmit = ieee802154:transmission({FrameControl, MacHeader, Frame}),
322   case Transmit of
323     {ok, _} ->
324       io:format("Packet sent successfully~n"),
325       From ! ok,
326       {next_state, idle, Data};
327     {error, Error} ->
328       io:format("Transmission error: ~p~n", [Error]),
329       From ! {error, Error},
330       {next_state, idle, Data}
331   end.
332
333 %----- Tx datagram state -----
334
335 %-----
336 %% @doc Handles the transmission of a datagram.
337 %% @spec tx_datagram(atom(), term(), map()) -> {next_state, atom(), map()}.
338 %-----
339 tx_datagram(internal, {tx_datagram}, Data) ->
340   #{data := {Ipv6Pckt, FrameControl, MacHeader, From}} = Data,
341   Transmit = ieee802154:transmission({FrameControl, MacHeader, <<?IPV6_DHTYPE:8,
      Ipv6Pckt/bitstring>>}),
342   case Transmit of
343     {ok, _} ->
344       From ! ok,
345       {next_state, idle, Data};
346     {error, Error} ->
347       From ! {error, Error},
348       {next_state, idle, Data}
349   end.
350
351 %----- Tx packet state -----
352
353 %-----
354 %% @doc Handles the transmission of a packet.

```

```

355 %% @spec tx_packet(atom(), term(), map()) -> {next_state, atom(), map()}.
356 %-----
357 tx_packet(internal, {tx_packet}, Data) ->
358   #{data := {Ipv6Pckt, PcktInfo, Extended_hopsleft, From},
359     node_mac_addr := CurrNodeMacAdd, seqNum := SeqNum, fragment_tag := Tag} =
360     Data,
361     DestAddress = PcktInfo#ipv6PckInfo.destAddress,
362     SrcAddress = PcktInfo#ipv6PckInfo.sourceAddress,
363     Payload = PcktInfo#ipv6PckInfo.payload,
364     DestMacAddress = lowpan_core:getEUI64MacAddr(DestAddress),
365     SenderMacAdd = lowpan_core:getEUI64MacAddr(SrcAddress),
366     io:format("Final destination: ~p~n", [DestMacAddress]),
367     io:format("Searching next hop...~n"),
368     {RouteExist, MeshedHdrBin, MH} = lowpan_core:getNextHop(CurrNodeMacAdd,
369       SenderMacAdd, DestMacAddress, DestAddress, SeqNum+1, Extended_hopsleft),
370     {CompressedHeader, _} = lowpan_core:compressIpv6Header(Ipv6Pckt, RouteExist),
371     CompressedPacket = <<CompressedHeader/binary, Payload/bitstring>>,
372     _CompressedPacketLen = byte_size(CompressedPacket),
373     {FragReq, Fragments} = lowpan_core:triggerFragmentation(CompressedPacket, Tag,
374       RouteExist),
375     FC = #frame_control{ack_req = ?ENABLED,
376       frame_type = ?FTYPE_DATA,
377       src_addr_mode = ?EXTENDED,
378       dest_addr_mode = ?EXTENDED},
379
380     case FragReq of
381     true ->
382       {Response, _NoAckCnt} = sendFragments(RouteExist, Fragments, 1,
383         MeshedHdrBin, MH, FC, Tag, 0),
384       NewTag = Tag+1 rem ?MAX_TAG_VALUE,
385       From ! Response,
386       {next_state, idle, Data#{fragments => Fragments, fragment_tag =>
387         NewTag}};
388     false ->
389       {Response, _NoAckCnt} = sendFragment(RouteExist, Fragments,
390         MeshedHdrBin, MH, FC, Tag),
391       NewTag = Tag+1 rem ?MAX_TAG_VALUE,
392       From ! Response,
393       {next_state, idle, Data#{fragments => Fragments, fragment_tag =>
394         NewTag}};
395     size_err ->
396       io:format("The datagram size exceed the authorized length~n"),
397       From ! error_frag_size,
398       {next_state, idle, Data}
399     end.
400 tx_packet_metrics(internal, {tx_packet_metrics}, Data) ->
401   #{data := {Ipv6Pckt, PcktInfo, Extended_hopsleft, From},
402     node_mac_addr := CurrNodeMacAdd, seqNum := SeqNum, metrics := Metrics,
403     fragment_tag := Tag} = Data,
404     DestAddress = PcktInfo#ipv6PckInfo.destAddress,
405     SrcAddress = PcktInfo#ipv6PckInfo.sourceAddress,
406     Payload = PcktInfo#ipv6PckInfo.payload,
407     DestMacAddress = lowpan_core:getEUI64MacAddr(DestAddress),
408     SenderMacAdd = lowpan_core:getEUI64MacAddr(SrcAddress),
409     PcktHeader = ipv6:getHeader(Ipv6Pckt),
410     io:format("Final destination: ~p~n", [DestMacAddress]),
411     io:format("Searching next hop...~n"),
412     {RouteExist, MeshedHdrBin, MH} = lowpan_core:getNextHop(CurrNodeMacAdd,
413       SenderMacAdd, DestMacAddress, DestAddress, SeqNum+1, Extended_hopsleft),
414     {CompressedHeader, _} = lowpan_core:compressIpv6Header(Ipv6Pckt, RouteExist),
415     CompressedPacket = <<CompressedHeader/binary, Payload/bitstring>>,
416     CompressedPacketLen = byte_size(CompressedPacket),

```

```

408     io:format("Compressed packet len: ~p bytes~n",[CompressedPacketLen]),
409     {FragReq, Fragments} = lowpan_core:triggerFragmentation(CompressedPacket, Tag,
RouteExist),
410     FC = #frame_control{ack_req = ?ENABLED,
411                       frame_type = ?FTYPE_DATA,
412                       src_addr_mode = ?EXTENDED,
413                       dest_addr_mode = ?EXTENDED},
414
415     case FragReq of
416     true ->
417         NewTag = Tag+1 rem ?MAX_TAG_VALUE,
418         StartTime = os:system_time(millisecond),
419
420         NewData = Data#{caller => From, fragment_tag => NewTag, ack_req =>
true},
421         set_nodeData_value(state_data, NewData),
422         {ok, NoAckCnt} = sendFragments(RouteExist, Fragments, 1, MeshedHdrBin,
MH, FC, Tag, 0),
423         FragmentsNbr = length(Fragments),
424         AckCounter = FragmentsNbr - NoAckCnt,
425         NewMetrics = Metrics#metrics{fragments_nbr = FragmentsNbr, ack_counter
= AckCounter, start_time = StartTime,
426                                     pkt_len = byte_size(PcktHeader),
                                     compressed_pkt_len = byte_size(
CompressedHeader)},
427         MetricsResult = {ok, NewMetrics},
428         ResetMetrics = Metrics#metrics{fragments_nbr = 0, ack_counter = 0,
start_time = 0,
429                                     pkt_len = 0, compressed_pkt_len = 0},
430
431         ResetData = Data#{caller => From, ack_req => false, metrics =>
ResetMetrics},
432         From ! MetricsResult,
433         {next_state, idle, ResetData#{fragments => Fragments, fragment_tag =>
NewTag}};
434
435     false ->
436         NewTag = Tag+1 rem ?MAX_TAG_VALUE,
437         StartTime = os:system_time(millisecond),
438         NewData = Data#{caller => From, fragment_tag => NewTag, ack_req =>
true},
439         set_nodeData_value(state_data, NewData),
440         {_R, NoAckCnt} = sendFragment(RouteExist, Fragments, MeshedHdrBin, MH,
FC, Tag),
441         FragmentsNbr = 1,
442         AckCounter = FragmentsNbr - NoAckCnt,
443         NewMetrics = Metrics#metrics{fragments_nbr = FragmentsNbr, ack_counter
= AckCounter, start_time = StartTime,
444                                     pkt_len = byte_size(PcktHeader),
                                     compressed_pkt_len = byte_size(
CompressedHeader)},
445         MetricsResult = {ok, NewMetrics},
446         ResetMetrics = Metrics#metrics{fragments_nbr = 0, ack_counter = 0,
start_time = 0,
447                                     pkt_len = 0, compressed_pkt_len = 0},
448         ResetData = Data#{caller => From, ack_req => false, metrics =>
ResetMetrics},
449         From ! MetricsResult,
450         {next_state, idle, ResetData#{fragments => Fragments, fragment_tag =>
NewTag}};
451     size_err ->
452         io:format("The datagram size exceed the authorized length~n"),

```



```

453         From ! error_frag_size,
454         {next_state, idle, Data}
455     end.
456
457 %----- Rx frame state -----
458
459 %-----
460 %% @doc Handles the reception of a frame.
461 %% @spec rx_frame(atom(), term(), map()) -> {next_state, atom(), map()} | {
462     keep_state, map()}.
463 %-----
464 rx_frame(internal, {rx_frame}, Data) ->
465     #{caller := From} = Data,
466     {keep_state, Data#{caller => From}};
467 rx_frame(cast, {frame_rx, _From}, Data) ->
468     {keep_state, Data};
469 rx_frame(cast, {new_frame_rx, IsMeshedPckt, OriginatorAddr, Datagram}, Data) ->
470     #{caller := From, node_mac_addr := CurrNodeMacAdd} = Data,
471     case Datagram of
472     <<?IPHC_DHTYPE:3, _Rest/bitstring>> ->
473         io:format("Received a compressed datagram, starting reassembly~n"),
474         From ! {reassembled_packet, IsMeshedPckt, OriginatorAddr,
475             CurrNodeMacAdd, Datagram},
476         {next_state, idle, Data};
477
478     <<?IPV6_DHTYPE:8, Payload/bitstring>> ->
479         io:format("Received a uncompressed IPv6 datagram, starting reassembly~
480             n"),
481         From ! {reassembled_packet, IsMeshedPckt, OriginatorAddr,
482             CurrNodeMacAdd, Payload},
483         {next_state, idle, Data};
484
485     <<Type:5, _Rest/bitstring>> when Type == ?FRAG1_DHTYPE; Type == ?
486         FRAGN_DHTYPE ->
487         FragInfo = lowpan_core:datagramInfo(Datagram),
488         Info = FragInfo#datagramInfo.datagramTag,
489         NewData = Data#{additional_info => Info},
490         io:format("Storing fragment~n"),
491         gen_statem:cast(?MODULE, {add_fragment, IsMeshedPckt, OriginatorAddr,
492             Datagram}),
493         {keep_state, NewData}
494     end;
495 rx_frame(cast, {add_fragment, IsMeshedPckt, OriginatorAddr, Datagram}, Data) ->
496     {next_state, collect, Data#{is_meshed_pckt => IsMeshedPckt, originator_addr =>
497         OriginatorAddr, datagram => Datagram},
498     [{next_event, internal, {start_collect}}]};
499 rx_frame(cast, {forward, Datagram, IsMeshedPckt, MeshPckInfo, FinalDstMacAdd,
500     CurrNodeMacAdd, FC, MH}, Data) ->
501     NewData = Data#{datagram => Datagram, is_meshed_pckt => IsMeshedPckt,
502         mesh_pck_info => MeshPckInfo, final_dst_mac_add =>
503             FinalDstMacAdd,
504         curr_node_mac_add => CurrNodeMacAdd, fc => FC, mh => MH},
505     {next_state, forward, NewData, [{next_event, internal, {start_forward}}]}.
506 %----- Rx new frame state -----
507

```

```

506
507 %----- Collect state -----
508
509 %-----
510 %% @doc Handles the collection of fragments.
511 %% @spec collect(atom(), term(), map()) -> {next_state, atom(), map()} | {
    keep_state, map()}.
512 %-----
513 collect(internal, {start_collect}, Data) ->
514   #{is_meshed_pckt := IsMeshedPckt, originator_addr := OriginatorAddr, datagram
    := Datagram,
515   datagram_map := DatagramMap, caller := From, node_mac_addr := CurrNodeMacAdd}
    = Data,
516
517   DtgInfo = lowpan_core:datagramInfo(Datagram),
518
519   Size = DtgInfo#datagramInfo.datagramSize,
520   Tag = DtgInfo#datagramInfo.datagramTag,
521   Offset = DtgInfo#datagramInfo.datagramOffset,
522   Payload = DtgInfo#datagramInfo.payload,
523
524   Key = {OriginatorAddr, Tag},
525   CurrTime = os:system_time(second),
526   case lowpan_core:storeFragment(DatagramMap, Key, Offset, Payload, CurrTime,
    Size, Tag, From) of
527     {complete_first_frag, ReassembledPacket} ->
528       io:format("Complete for pckt ~p~n", [Key]),
529       From ! {reassembled_packet, IsMeshedPckt, OriginatorAddr,
    CurrNodeMacAdd, ReassembledPacket},
530       {next_state, idle, Data};
531
532     {complete, UpdatedDatagram} ->
533       gen_statem:cast(?MODULE, {complete, IsMeshedPckt, OriginatorAddr, Key,
    UpdatedDatagram}),
534       NewData = Data#{key => Key},
535       {keep_state, NewData};
536
537     {duplicate, _} ->
538       io:format("Duplicate frame detected~n"),
539       NewData = Data#{key => Key},
540       {next_state, rx_frame, NewData};
541
542     {incomplete_first, EntryKey} ->
543       io:format("Incomplete first datagram, waiting for other fragments ~n")
544       ,
545       erlang:send_after(?REASSEMBLY_TIMEOUT, From, {reassemble_timeout,
    DatagramMap, EntryKey}),
546       NewData = Data#{key => Key},
547       {next_state, rx_frame, NewData};
548
549     {incomplete, _} ->
550       io:format("Incomplete datagram, waiting for other fragments ~n"),
551       NewData = Data#{key => Key},
552       {next_state, rx_frame, NewData}
553   end;
554 collect(cast, {complete, IsMeshedPckt, OriginatorAddr, Key, UpdatedDatagram}, Data
) ->
555   NewData = Data#{is_meshed_pckt => IsMeshedPckt, originator_addr =>
    OriginatorAddr,
556   key => Key, updated_datagram => UpdatedDatagram},

```

```

557     {next_state, reassemble, NewData, [{next_event, internal, {start_reassemble}}]}
558     }.
559 %----- Reassembly state -----
560
561 %-----
562 %% @doc Handles the reassembly of fragments.
563 %% @spec reassemble(atom(), term(), map()) -> {next_state, atom(), map()}.
564 %-----
565 reassemble(internal, {start_reassemble}, Data) ->
566     %io:format("Data: ~p~n", [Data]),
567     #{datagram_map := DatagramMap, caller := From, additional_info:=Info,
568       node_mac_addr := CurrNodeMacAdd,
569       is_meshed_pckt := IsMeshedPckt, originator_addr := OriginatorAddr,
570       key := Key, updated_datagram := UpdatedDatagram} = Data,
571
572     ReassembledPacket = lowpan_core:reassemble(UpdatedDatagram),
573     io:format("Complete for pckt ~p~n", [Key]),
574     ets:delete(DatagramMap, Key),
575     case Info of
576     ?INFO_ON ->
577         From ! {additional_info, Info, ReassembledPacket};
578     - ->
579         From ! {reassembled_packet, IsMeshedPckt, OriginatorAddr,
580               CurrNodeMacAdd, ReassembledPacket}
581     end,
582     {next_state, idle, Data}.
583
584 %----- Forward state -----
585
586 %-----
587 %% @doc Handles the forwarding of datagrams.
588 %% @spec forward(atom(), term(), map()) -> {next_state, atom(), map()}.
589 %-----
590 forward(internal, {start_forward}, Data) ->
591     #{datagram := Datagram, is_meshed_pckt := IsMeshedPckt,
592       mesh_pck_info := MeshPckInfo, final_dst_mac_add :=
593         FinalDstMacAdd,
594       curr_node_mac_add := CurrNodeMacAdd, fc := FC, mh := MH} =
595     Data,
596
597     NewDatagram =
598     case IsMeshedPckt of
599     true ->
600         update_datagram(MeshPckInfo, Datagram, Data);
601     false ->
602         SenderMacAdd = MH#mac_header.src_addr,
603         lowpan_core:createNewMeshDatagram(Datagram, SenderMacAdd,
604           FinalDstMacAdd)
605     end,
606
607     case NewDatagram of
608     {discard, _} ->
609         {next_state, rx_frame, Data};
610     - ->
611         DestMacAddress = lowpan_core:convertAddrToBin(FinalDstMacAdd),
612         io:format("Searching next hop in the routing table...~n"),
613         NextHopAddr = routing_table:getRoute(DestMacAddress),
614
615         case NextHopAddr of
616         DestMacAddress ->
617             io:format("Direct link found~nForwarding to node: ~p~n", [
618               NextHopAddr]);
619         _ ->

```

```

612             io:format("Next hop found~nForwarding to node: ~p~n", [
613                 NextHopAddr])
614         end,
615         NewMH = MH#mac_header{src_addr = CurrNodeMacAdd, dest_addr =
616             NextHopAddr},
617         io:format("-----~n"),
618         forward_datagram(NewDatagram, FC, NewMH, Data)
619     end.
620
621 %----- Utility functions -----
622 %% @doc Sends a fragment
623 %% @spec sendFragment(boolean(), binary(), binary(), map(), term(), integer()) ->
624 %% {ok, integer()} | {Error, integer()}.
625 sendFragment(RouteExist, CompressedPacket, MeshedHdrBin, MH, FC, Tag) ->
626     Pckt = case RouteExist of
627         true ->
628             <<MeshedHdrBin/binary, CompressedPacket/bitstring>>;
629         false ->
630             CompressedPacket
631     end,
632     MacHeader = MH#mac_header{seqnum = Tag},
633     case ieee802154:transmission({FC, MacHeader, Pckt}) of
634     {ok, _} ->
635         io:format("-p-byte packet successfully sent~n", [ byte_size(Pckt)]),
636         {ok, 0};
637     {error, Error} ->
638         io:format("Transmission error: ~p~n", [Error]),
639         NoAck = 1,
640         {Error, NoAck}
641     end.
642
643 %-----
644 %% @doc Sends list of fragments
645 %% @spec sendFragments(boolean(), list(), integer(), binary(), map(), term(),
646 %% integer(), integer()) -> {ok, integer()}.
647 sendFragments(RouteExist, [{FragHeader, FragPayload} | Rest], PcktCounter,
648     MeshedHdrBin, MH, FC, Tag, NoAckCnt) ->
649     Pckt = case RouteExist of
650         true ->
651             <<MeshedHdrBin/binary, FragHeader/binary, FragPayload/
652                 bitstring>>;
653         false ->
654             <<FragHeader/binary, FragPayload/bitstring>>
655     end,
656     MacHeader = MH#mac_header{seqnum = Tag+PcktCounter},
657     case ieee802154:transmission({FC, MacHeader, Pckt}) of
658     {ok, _} ->
659         io:format("-pth fragment: ~p bytes sent~n", [PcktCounter, byte_size(
660             Pckt)]),
661         sendFragments(RouteExist, Rest, PcktCounter + 1, MeshedHdrBin,
662             MacHeader, FC, Tag, NoAckCnt);
663     {error, Error} ->
664         io:format("Error during transmission of fragment ~p: ~p~n", [
665             PcktCounter, Error]),
666         sendFragments(RouteExist, Rest, PcktCounter+1, MeshedHdrBin, MacHeader
667             , FC, Tag, NoAckCnt + 1)
668     end;

```

```

663 sendFragments(_RouteExist, [], _PcktCounter, _MeshedHdrBin, _MH, _FC, _Tag,
NoAckCnt) ->
664     case NoAckCnt of
665         0 ->
666             io:format("Packet successfully sent~n");
667         _->
668             io:format("Issue during transmission~n")
669     end,
670     {ok, NoAckCnt}.
671
672 %-----
673 %% @doc Updates the datagram with new mesh header information.
674 %% @spec update_datagram(map(), binary(), map()) -> binary() | {discard, term()}.
675 %-----
676 update_datagram(MeshInfo, Datagram, Data) ->
677     HopsLeft = MeshInfo#meshInfo.hops_left,
678
679     {Is_Extended_hopsleft, HopLft} =
680         case HopsLeft of
681             ?DeepHopsLeft ->
682                 HopsLft = MeshInfo#meshInfo.deep_hops_left-1,
683                 {true, HopsLft};
684             _ -> HopsLft = HopsLeft-1,
685                 {false, HopsLft}
686         end,
687
688     case {Is_Extended_hopsleft, HopLft} of
689         {_, 0} ->
690             {discard, discard_datagram(Datagram, Data)};
691         {false, _} ->
692             Payload = MeshInfo#meshInfo.payload,
693             MeshHeader =
694                 #mesh_header{v_bit = MeshInfo#meshInfo.v_bit,
695                             f_bit = MeshInfo#meshInfo.f_bit,
696                             hops_left = HopsLft,
697                             originator_address = MeshInfo#meshInfo.
698                                 originator_address,
699                             final_destination_address = MeshInfo#meshInfo.
700                                 final_destination_address},
701
702             BinMeshHeader = lowpan_core:buildMeshHeader(MeshHeader),
703             <<BinMeshHeader/binary, Payload/bitstring>>;
704         {true, _} ->
705             Payload = MeshInfo#meshInfo.payload,
706             VBit = MeshInfo#meshInfo.v_bit,
707             FBit = MeshInfo#meshInfo.f_bit,
708             OriginatorAddress = MeshInfo#meshInfo.originator_address,
709             FinalDestinationAddress = MeshInfo#meshInfo.final_destination_address
710             ,
711             BinMeshHeader = <<?MESH_DHTYPE:2, VBit:1, FBit:1, ?DeepHopsLeft:4,
712                 OriginatorAddress/binary, FinalDestinationAddress/
713                 binary, HopLft:8>>,
714             <<BinMeshHeader/binary, Payload/bitstring>>
715     end.
716
717 %-----
718 %% @doc Discards the datagram when hop count reaches zero.
719 %% @spec discard_datagram(binary(), map()) -> {next_state, atom(), map()}.
720 %-----
721 discard_datagram(_, Data = #{caller := From})->
722     io:format("Hop left value: 0, discarding the datagram~n"),

```

```

720     From ! dtg_discarded,
721     {next_state, rx_frame, Data}.
722
723 %-----
724 %% @doc Forwards a datagram to the next hop.
725 %% @spec forward_datagram(binary(), term(), map(), map()) -> {next_state, atom(),
726 %% @spec forward_datagram(binary(), term(), map(), map()) -> {next_state, atom(),
727 %% @spec forward_datagram(binary(), term(), map(), map()) -> {next_state, atom(),
728 forward_datagram(Frame, FrameControl, MacHeader, Data = #{caller := From}) ->
729     case Frame of
730     <<?NALP_DHTYPE, _/bitstring>> ->
731         io:format("The received frame is not a lowpan frame~n"),
732         From ! error_nalp;
733     _->
734         Transmit = ieee802154:transmission({FrameControl, MacHeader, Frame}),
735         case Transmit of
736         {ok, _} ->
737             io:format("Packet sent successfully~n");
738         {error, Error} ->
739             io:format("Transmission error: ~p~n", [Error])
740         end
741     end,
742     io:format("-----~n"),
743     {next_state, rx_frame, Data}.
744
745 handle_ack(Metrics) ->
746     TotalFragments = Metrics#metrics.fragments_nbr,
747     AckCounter = Metrics#metrics.ack_counter,
748     EndTime = os:system_time(millisecond),
749
750     RTT = EndTime - Metrics#metrics.start_time,
751     SuccessRate = AckCounter / TotalFragments,
752     _LossRate = 1 - SuccessRate,
753
754     OrigPcktLen = Metrics#metrics.pckt_len,
755     CompPcktLen = Metrics#metrics.compressed_pckt_len,
756     CompressionRatio = (CompPcktLen/OrigPcktLen),
757     {ok, RTT, SuccessRate, CompressionRatio}.
758
759 callback_mode() ->
760     [state_functions].
761
762 %-----
763 %% @doc Sets up ETS table for node information.
764 %% @spec setup_node_info_ets() -> atom().
765 %% @spec setup_node_info_ets() -> atom().
766 %% @spec setup_node_info_ets() -> atom().
767
768 setup_node_info_ets() ->
769     ets:new(nodeData, [named_table, public, {keypos, 1}]).
770
771 %-----
772 %% @doc Sets a value in the node data ETS table.
773 %% @spec set_nodeData_value(term(), term()) -> ok.
774 %% @spec set_nodeData_value(term(), term()) -> ok.
775 %% @spec set_nodeData_value(term(), term()) -> ok.
776
777 set_nodeData_value(Key, Value) ->
778     ets:insert(nodeData, {Key, Value}).
779
780 %-----
781 %% @doc Retrieves a value from the node data ETS table.
782 %% @spec get_nodeData_value(term()) -> term() | undefined.
783 %% @spec get_nodeData_value(term()) -> term() | undefined.
784 %% @spec get_nodeData_value(term()) -> term() | undefined.
785
786 get_nodeData_value(Key) ->
787     case ets:lookup(nodeData, Key) of
788     [] ->

```

```

781         undefined;
782         [{_, Value}] ->
783             Value
784     end.
785
786 %-----
787 %% @doc Sets up the IEEE 802.15.4 layer.
788 %% @spec ieee802154_setup(binary()) -> ok.
789 %-----
790 ieee802154_setup(MacAddr)->
791     ieee802154:start(#ieee_parameters{
792         phy_layer = mock_phy_network, % uncomment when testing
793         duty_cycle = duty_cycle_non_beacon,
794         input_callback = fun lowpan_api:inputCallback/4
795     }),
796
797     case application:get_env(robot, pan_id) of
798     {ok, PanId} ->
799         ieee802154:set_pib_attribute(mac_pan_id, PanId);
800     _ ->
801         ok
802     end,
803
804     case byte_size(MacAddr) of
805     ?EXTENDED_ADDR_LEN -> ieee802154:set_pib_attribute(mac_extended_address,
806         MacAddr);
807     ?SHORT_ADDR_LEN -> ieee802154:set_pib_attribute(mac_short_address, MacAddr
808     )
809     end,
810
811     ieee802154:rx_on(),
812     io:format("-p: IEEE 802.15.4 layer successfully launched ~n",[node()]).

```

A.6 Routing table code

```

1 -module(routing_table).
2
3 -behaviour(gen_server).
4
5 %%% API
6 -export([
7     start_link/1,
8     stop/0,
9     addRoute/2,
10    deleteRoute/1,
11    getRoute/1,
12    updateRoute/2,
13    resetRouting_table/0
14 ]).
15
16 %%% gen_server callbacks
17 -export([init/1, handle_call/3, handle_cast/2, terminate/2, code_change/3]).
18
19 %%% API functions
20
21 start_link(RoutingTable) ->
22     gen_server:start_link({local, ?MODULE}, ?MODULE, RoutingTable, []).

```

```

23
24 stop() ->
25     gen_server:stop(?MODULE).
26
27 addRoute(DestAddr, NextHAddr) ->
28     gen_server:call(?MODULE, {add_route, DestAddr, NextHAddr}).
29
30 deleteRoute(DestAddr) ->
31     gen_server:call(?MODULE, {delete_route, DestAddr}).
32
33 getRoute(DestAddr) ->
34     gen_server:call(?MODULE, {get_route, DestAddr}).
35
36 updateRoute(DestAddr, NextHAddr) ->
37     gen_server:call(?MODULE, {update_route, DestAddr, NextHAddr}).
38
39 resetRouting_table() ->
40     gen_server:call(?MODULE, reset).
41
42 %% gen_server callbacks
43 init(RoutingTable) ->
44     {ok, RoutingTable}.
45
46
47 handle_call({add_route, DestAddr, NextHAddr}, _From, RoutingTable) ->
48     NewTable = maps:put(DestAddr, NextHAddr, RoutingTable),
49     {reply, ok, NewTable};
50
51 handle_call({delete_route, DestAddr}, _From, RoutingTable) ->
52     NewTable = maps:remove(DestAddr, RoutingTable),
53     {reply, ok, NewTable};
54
55 handle_call({get_route, DestAddr}, _From, RoutingTable) ->
56     NextHAddr = maps:get(DestAddr, RoutingTable, undefined),
57     {reply, NextHAddr, RoutingTable};
58
59 handle_call({update_route, DestAddr, NextHAddr}, _From, RoutingTable) ->
60     NewTable = maps:put(DestAddr, NextHAddr, RoutingTable),
61     {reply, ok, NewTable};
62
63 handle_call(reset, _From, _MapState) ->
64     {reply, ok, #{}}.
65
66 handle_cast(_, State) ->
67     {noreply, State}.
68
69 terminate(_Reason, _State) ->
70     ok.
71
72 code_change(_OldVsn, State, _Extra) ->
73     {ok, State}.

```

A.7 Lowpan ipv6 code

```

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

```



```

4
5 -record(ipv6_header, {
6     version = 2#0110, % 4-bit (version 6)
7     traffic_class, % 8-bit
8     flow_label, % 20-bit
9     payload_length, % 16-bit
10    next_header, % 8-bit
11    hop_limit, % 8-bit
12    source_address, % 128-bit
13    destination_address % 128-bit
14 }).
15
16 -record(udp_header, {
17     source_port, % 16-bit identifies the sender's port
18     destination_port, % 16-bit identifies the receiver's port and is required
19     length, % 16-bit indicates the length in bytes of the UDP datagram
20     checksum % 16-bit may be used for error-checking of the header and data
21 }).
22
23 %-----
24 %% @doc Returns an IPv6 header in binary format
25 %% @spec build_ipv6_header(#ipv6_header{}) -> binary().
26 %-----
27 -spec buildIpv6Header(#ipv6_header{}) -> binary().
28 buildIpv6Header(IPv6Header) ->
29     #ipv6_header{
30         version = _Version,
31         traffic_class = Traffic_class,
32         flow_label = Flow_label,
33         payload_length = Payload_length,
34         next_header = Next_header,
35         hop_limit = Hop_limit,
36         source_address = SourceAddr,
37         destination_address = DestAddr
38     } = IPv6Header,
39
40     <<6:4,Traffic_class:8,Flow_label:20,Payload_length:16,Next_header:8,Hop_limit
41         :8,SourceAddr/binary, DestAddr/binary>>.
42
43 %-----
44 %% @doc Extracts the IPv6 header from a packet
45 %% @spec get_header(binary()) -> binary().
46 %-----
47 -spec getHeader(binary()) -> binary().
48 getHeader(Ipv6Pckt) ->
49     <<Header:320, _/bitstring>> = Ipv6Pckt,
50     <<Header:320>>.
51
52 %-----
53 %% @doc Returns a UDP header in binary format
54 %% @spec build_udp_header(#udp_header{}) -> binary().
55 %-----
56 -spec buildUdpHeader(#udp_header{}) -> binary().
57 buildUdpHeader(UdpHeader) ->
58     #udp_header{
59         source_port = SourcePort,
60         destination_port = DestinationPort,
61         length = Length,
62         checksum = Checksum
63     } = UdpHeader,
64
65     <<SourcePort:16, DestinationPort:16, Length:16, Checksum:16>>.

```

```

65
66 %-----
67 %% @doc Builds an IPv6 packet with the given header and payload
68 %% @spec buildIpv6Packet(#ipv6_header{}, binary()) -> binary().
69 %-----
70 -spec buildIpv6Packet(#ipv6_header{}, binary()) -> binary().
71 buildIpv6Packet(IPv6Header, Payload) ->
72     Header = buildIpv6Header(IPv6Header),
73     IPv6Packet = <<Header/binary, Payload/bitstring>>,
74     IPv6Packet.
75
76 %-----
77 %% @doc Builds an IPv6 packet with the given IPv6 header, UDP header, and payload
78 %% @spec build_ipv6_udp_packet(#ipv6_header{}, #udp_header{}, binary()) -> binary
79 %-----
80 -spec buildIpv6UdpPacket(#ipv6_header{}, #udp_header{}, binary()) -> binary().
81 buildIpv6UdpPacket(IPv6Header, UdpHeader, Payload) ->
82     IpHeader = buildIpv6Header(IPv6Header),
83     UdpH = buildUdpHeader(UdpHeader),
84     IPv6Packet = <<IpHeader/binary, UdpH/binary, Payload/bitstring>>,
85     IPv6Packet.

```

A.8 Utils file for testing code

```

1 -include("lowpan.hrl").
2
3 %-----
4 % Common value for testing purpose
5 %-----
6
7 -define(Payload, <<"Hello world this is an ipv6 packet for testing purpose">>).
8 -define(BigPayload, lowpan_core:generateChunks()).
9 -define(PayloadLength, byte_size(?Payload)).
10
11 -define(Node1Address, lowpan_core:generateLLAddr(?Node1MacAddress)). % generates a
12     link local address based on the mac address
13 -define(Node2Address, lowpan_core:generateLLAddr(?Node2MacAddress)).
14 -define(Node3Address, lowpan_core:generateLLAddr(?Node3MacAddress)).
15 -define(Node4Address, lowpan_core:generateLLAddr(?Node4MacAddress)).
16 -define(Node5Address, lowpan_core:generateLLAddr(?Node5MacAddress)).
17
18 -define(IPv6Header, #ipv6_header{
19     version = 6,
20     traffic_class = 0,
21     flow_label = 0,
22     payload_length = ?PayloadLength,
23     next_header = 12,
24     hop_limit = 64,
25     source_address = ?Node1Address,
26     destination_address = ?Node2Address
27 }).
28
29 -define(IPv6Header3, #ipv6_header{
30     version = 6,
31     traffic_class = 0,
32     flow_label = 0,

```

```

32     payload_length = ?PayloadLength,
33     next_header = 12,
34     hop_limit = 64,
35     source_address = ?Node1Address,
36     destination_address = ?Node3Address
37 }).
38
39 -define(IPv6Header4, #ipv6_header{
40     version = 6,
41     traffic_class = 0,
42     flow_label = 0,
43     payload_length = ?PayloadLength,
44     next_header = 12,
45     hop_limit = 64,
46     source_address = ?Node1Address,
47     destination_address = ?Node4Address
48 }).
49
50 -define(IPv6Header5, #ipv6_header{
51     version = 6,
52     traffic_class = 0,
53     flow_label = 0,
54     payload_length = ?PayloadLength,
55     next_header = 12,
56     hop_limit = 64,
57     source_address = ?Node1Address,
58     destination_address = ?Node5Address
59 }).
60
61 -define(FrameControl, #frame_control{
62     frame_type = ?FTYPE_DATA,
63     src_addr_mode = ?EXTENDED,
64     dest_addr_mode = ?EXTENDED
65 }).
66 -define(Ipv6Pckt, ipv6:buildIpv6Packet(?IPv6Header, ?Payload)).
67 -define(MacHeader, #mac_header{src_addr = ?Node1MacAddress, dest_addr = ?
    Node2MacAddress}).
68
69
70 %-----
71 % multiple hop Routing tables
72 %-----
73
74 -define(Node1_multiple_hop_routing_table,
75     #{?node4_addr => ?node2_addr}).
76
77 -define(Node2_multiple_hop_routing_table,
78     #{?node4_addr => ?node3_addr}).
79
80 -define(Node3_multiple_hop_routing_table,
81     #{?node4_addr => ?node4_addr}).
82
83 -define(Node4_multiple_hop_routing_table,
84     #{?node4_addr => ?node4_addr,
85     ?node3_addr => ?node3_addr}).

```

A.9 Functional testing code

```

1 -module(lowpan_test_SUITE).
2
3 -include("../src/utils.hrl").
4
5 -export([all/0, init_per_testcase/1, end_per_testcase/1]).
6 -export([
7     pkt_encapsulation_test/1, fragmentation_test/1, datagram_info_test/1,
8     reassemble_fragments_list_test/1, reassemble_single_fragments_test/1,
9     reassemble_full_ipv6_pkt_test/1, compress_header_example1_test/1,
10    compress_header_example2_test/1, link_local_addr_pkt_comp/1,
11    multicast_addr_pkt_comp/1, global_context_pkt_comp1/1, udp_nh_pkt_comp/1,
12    tcp_nh_pkt_comp/1, icmp_nh_pkt_comp/1, unc_ipv6/1, iphc_pkt_16bit_addr/1,
13    iphc_pkt_64bit_addr/1, msh_pkt/1, extended_EUI64_from_64mac/1,
14    extended_EUI64_from_48mac/1,
15    extended_EUI64_from_16mac/1, check_tag_unicity/1, link_local_from_16mac/1,
16    multicast_addr_validity/1,
17    broadcast_pkt/1
18 ]).
19 % -export([cooja_example3/1]).
20 % -export([cooja_example2/1]).
21 % -export([cooja_example1/1]).
22
23 all() ->
24     [
25         pkt_encapsulation_test,
26         datagram_info_test,
27         reassemble_fragments_list_test,
28         reassemble_single_fragments_test,
29         reassemble_full_ipv6_pkt_test,
30         compress_header_example1_test,
31         compress_header_example2_test,
32         link_local_addr_pkt_comp,
33         multicast_addr_pkt_comp,
34         global_context_pkt_comp1,
35         udp_nh_pkt_comp,
36         tcp_nh_pkt_comp,
37         icmp_nh_pkt_comp,
38         unc_ipv6,
39         iphc_pkt_64bit_addr,
40         iphc_pkt_16bit_addr,
41         msh_pkt, extended_EUI64_from_64mac, extended_EUI64_from_48mac,
42         extended_EUI64_from_16mac, check_tag_unicity, link_local_from_16mac,
43         multicast_addr_validity, broadcast_pkt
44     ].
45
46 init_per_testcase(Config) ->
47     Config.
48
49 end_per_testcase(_Config) ->
50     ok.
51
52 %-----
53 %                               6LoWPAN Packet Encapsulation
54 %-----
55
56 pkt_encapsulation_test(_Config) ->
57     Payload = <<"This is an Ipv6 pkt">>,
58     IPv6Header =
59     #ipv6_header{

```

```

60         version = 6,
61         traffic_class = 0,
62         flow_label = 0,
63         payload_length = byte_size(Payload),
64         next_header = 17,
65         hop_limit = 64,
66         source_address = <<1>>,
67         destination_address = <<2>>
68     },
69     IPv6Packet = ipv6:buildIPv6Packet(IPv6Header, Payload),
70     DhTypebinary = <<?IPv6_DHTYPE:8, 0:16>>,
71     ToCheck = <<DhTypebinary/binary, IPv6Packet/binary>>,
72     ToCheck = lowpan_core:pktEncapsulation(IPv6Header, Payload),
73     ok.
74
75
76 unc_ipv6(_Config) ->
77     Ipv6Pckt = ipv6:buildIPv6Packet(?IPv6Header, ?Payload),
78
79     Expected = <<?IPv6_DHTYPE:8, Ipv6Pckt/bitstring>>,
80     Expected = lowpan_core:getUncIpv6(Ipv6Pckt).
81
82 iphc_pckt_16bit_addr(_Config) ->
83     Node1Addr = lowpan_core:generateLLAddr(<<16#0001:16>>),
84     Node2Addr = lowpan_core:generateLLAddr(<<16#0002:16>>),
85     IPv6Header =
86         #ipv6_header{
87             version = 6,
88             traffic_class = 0,
89             flow_label = 0,
90             payload_length = byte_size(?Payload),
91             next_header = 12,
92             hop_limit = 64,
93             source_address = Node1Addr,
94             destination_address = Node2Addr
95         },
96     Ipv6Pckt = ipv6:buildIPv6Packet(IPv6Header,?Payload),
97
98     InlineData = <<12:8>>,
99     ExpectedHeader =
100         <<?IPHC_DHTYPE:3, 3:2, 0:1, 2:2, 0:1, 0:1, 3:2, 0:1, 0:1, 3:2, InlineData/
            binary>>,
101
102     % Create the IPHC packet
103     {IPHC, _} = lowpan_core:compressIPv6Header(Ipv6Pckt, false),
104     io:format("IPHC: ~p~n", [IPHC]),
105     io:format("ExpectedHeader: ~p~n", [ExpectedHeader]),
106     IPHC = ExpectedHeader.
107
108 iphc_pckt_64bit_addr(_Config) ->
109     InlineData = <<12:8, (?node1_addr)/binary, (?node2_addr)/binary>>,
110     ExpectedHeader =
111         <<?IPHC_DHTYPE:3, 3:2, 0:1, 2:2, 0:1, 0:1, 1:2, 0:1, 0:1, 1:2, InlineData/
            binary>>,
112
113     % Create the IPHC packet
114     {IPHC, _} = lowpan_core:compressIPv6Header(?Ipv6Pckt, false),
115     io:format("IPHC: ~p~n", [IPHC]),
116     io:format("ExpectedHeader: ~p~n", [ExpectedHeader]),
117     IPHC = ExpectedHeader.
118
119 msh_pckt(_Config) ->

```

```

120 MeshHeader =
121     #mesh_header{
122         v_bit = 0,
123         f_bit = 0,
124         hops_left = 14,
125         originator_address = ?Node1MacAddress,
126         final_destination_address = ?Node2MacAddress
127     },
128
129 BinMeshHeader = lowpan_core:buildMeshHeader(MeshHeader),
130 ExpectedHeader =
131     <<?MESH_DHTYPE:2, 0:1, 0:1, 14:4, ?Node1MacAddress/binary, ?
132         Node2MacAddress/binary>>,
133
134 ExpectedHeader = BinMeshHeader.
135
136 broadcast_pkt(_Config) ->
137     DestMacAddr = lowpan_core:generateEUI64MacAddr(<<16#1234:16>>),
138     MeshHeader =
139         #mesh_header{
140             v_bit = 0,
141             f_bit = 0,
142             hops_left = 14,
143             originator_address = ?Node1MacAddress,
144             final_destination_address = DestMacAddr
145         },
146
147     BinMeshHeader = lowpan_core:buildMeshHeader(MeshHeader),
148
149     DestAddr = <<16#FF02:16, 0:64, 1:16, 16#FF00:16, 16#1234:16>>,
150     DestAddress = binary:decode_unsigned(DestAddr),
151     {_, BroadcastHeader, _} = lowpan_core:getNextHop(?Node1MacAddress, ?
152         Node1MacAddress, DestMacAddr, DestAddress, 3, false),
153
154     ExpectedHeader = <<BinMeshHeader/bitstring, ?BCO_DHTYPE, 3:8>>,
155
156     io:format("Expected: ~p~n~nReceived: ~p~n", [ExpectedHeader, BroadcastHeader])
157
158     ExpectedHeader = BroadcastHeader.
159
160 %-----
161 %                               Ipv6 Packet Compression
162 %-----
163
164 %--- Basic IPHC test case
165
166 % Link-local address
167 link_local_addr_pkt_comp(_Config) ->
168     Payload = <<"Testing basic IPHC compression with link-local address">>,
169     IPv6Header =
170         #ipv6_header{
171             version = 6,
172             traffic_class = 0,
173             flow_label = 0,
174             payload_length = byte_size(Payload),
175             next_header = 0,
176             hop_limit = 64,
177             source_address = <<16#FE80:16, 0:48, ?Node1MacAddress/binary>>,
178             destination_address = <<16#FE80:16, 0:48, ?Node2MacAddress/binary>>
179         },
180
181     Ipv6Pckt = ipv6:buildIpv6Packet(IPv6Header, Payload),

```

```

179
180     Tf = 2#11,
181     Nh = 0,
182     Hlim = 2#10,
183     Cid = 0,
184     Sac = 0,
185     Sam = 2#01,
186     M = 0,
187     Dac = 0,
188     Dam = 2#01,
189     ExpectedCarriedInline =
190         #{
191             "SAM" => <<?Node1MacAddress/binary>>,
192             "DAM" => <<?Node2MacAddress/binary>>,
193             "NextHeader" => 0
194         },
195
196     InlineData =
197         <<0:8, ?Node1MacAddress/binary,
198             ?Node2MacAddress/binary>>,
199     ExpectedHeader =
200         <<?IPHC_DHTYPE:3, Tf:2, Nh:1, Hlim:2, Cid:1, Sac:1, Sam:2, M:1, Dac:1, Dam
201             :2, InlineData/binary>>,
202
203     {CompressedHeader, CarriedInlineData} = lowpan_core:compressIpv6Header(
204         Ipv6Pckt, false),
205     io:format("Expected ~p~nReceived ~p~n", [ExpectedHeader, CompressedHeader]),
206     ExpectedHeader = CompressedHeader,
207
208     ExpectedCarriedInline = CarriedInlineData,
209     ok.
210
211 % Multicast address
212 multicast_addr_pkt_comp(_Config) ->
213     Payload = <<"Testing basic IPHC compression with multicast address">>,
214     IPv6Header =
215         #ipv6_header{
216             version = 6,
217             traffic_class = 0,
218             flow_label = 2,
219             payload_length = byte_size(Payload),
220             %UDP
221             next_header = 0,
222             hop_limit = 1,
223             source_address = <<16#FE80:16, 0:48, ?Node1MacAddress/binary>>,
224             destination_address = <<16#FF02:16, 0:48, ?Node2MacAddress/binary>>
225         },
226
227     Ipv6Pckt = ipv6:buildIpv6Packet(IPv6Header, Payload),
228
229     Tf = 2#01,
230     Nh = 0,
231     Hlim = 2#01,
232     Cid = 0,
233     Sac = 0,
234     Sam = 2#01,
235     M = 1,
236     Dac = 0,
237     Dam = 2#00,
238
239     Dest = IPv6Header#ipv6_header.destination_address,
240     ExpectedCarriedInline =

```

```

239     #{
240         "SAM" => <<?Node1MacAddress/binary>>,
241         "DAM" => <<Dest/binary>>,
242         "NextHeader" => 0,
243         "ECN" => 0,
244         "FlowLabel" => 2
245     },
246
247
248     InlineData =
249         <<0:2, 0:2, 2:20, 0:8, ?Node1MacAddress/binary,
250             Dest/binary>>,
251
252     ExpectedHeader =
253         <<?IPHC_DHTYPE:3, Tf:2, Nh:1, Hlim:2, Cid:1, Sac:1, Sam:2, M:1, Dac:1, Dam
254             :2, InlineData/binary>>,
255
256     {CompressedHeader, CarriedInlineData} = lowpan_core:compressIpv6Header(
257         Ipv6Pckt, false),
258     io:format("Expected ~p~nReceived ~p~n", [ExpectedHeader, CompressedHeader]),
259     ExpectedHeader = CompressedHeader,
260
261     ExpectedCarriedInline = CarriedInlineData,
262     ok.
263
264 %---Global contexts test case, affected fields are cid, sac and dac
265 global_context_pkt_comp1(_Config) ->
266     Payload = <<"Testing basic IPHC compression with multicast address">>,
267     Source_address = <<16#2001:16, 0:48, ?Node1MacAddress/binary>>,
268     Destination_address = <<16#2001:16, 0:48, ?Node2MacAddress/binary>>,
269     Ipv6Header =
270         #ipv6_header{
271             version = 6,
272             traffic_class = 0,
273             flow_label = 3,
274             payload_length = byte_size(Payload),
275             %UDP
276             next_header = 0,
277             hop_limit = 255,
278             source_address = Source_address,
279             destination_address = Destination_address
280         },
281
282     Ipv6Pckt = ipv6:buildIpv6Packet(Ipv6Header, Payload),
283
284     Tf = 2#01,
285     Nh = 0,
286     Hlim = 2#11,
287     Cid = 0,
288     Sac = 1,
289     Sam = 2#00,
290     M = 0,
291     Dac = 1,
292     Dam = 2#00,
293
294     ExpectedCarriedInline =
295         #{
296             "SAM" => Source_address,
297             "NextHeader" => 0,
298             "ECN" => 0,
299             "FlowLabel" => 3,
300             "DAM" => Destination_address

```



```

299     },
300     io:format("ExpectedCarriedInline: ~p~n", [ExpectedCarriedInline]),
301
302     InlineData =
303         <<0:2, 0:2, 3:20, 0:8, Source_address/binary, Destination_address/binary
304         >>,
305     ExpectedHeader =
306         <<?IPHC_DHTYPE:3, Tf:2, Nh:1, Hlim:2, Cid:1, Sac:1, Sam:2, M:1, Dac:1, Dam
307         :2, InlineData/binary>>,
308
309     {CompressedHeader, CarriedInlineData} = lowpan_core:compressIpv6Header(
310         Ipv6Pckt, false),
311     io:format("Expected ~p~nReceived ~p~n", [ExpectedHeader, CompressedHeader]),
312     ExpectedHeader = CompressedHeader,
313
314     ExpectedCarriedInline = CarriedInlineData,
315     ok.
316
317 %---Different types of Next Headers test case
318 udp_nh_pckt_comp(_Config) ->
319     Payload = <<"Testing basic IPHC compression with link-local address">>,
320
321     PayloadLength = byte_size(Payload),
322     Source_address = <<16#FE80:16, 0:48, ?Node1MacAddress/binary>>,
323     Destination_address = <<16#FE80:16, 0:48, ?Node2MacAddress/binary>>,
324
325     UdpPckt = <<1025:16, 61617:16, 25:16, 16#f88c:16>>,
326
327     Ipv6Pckt =
328         <<6:4, 0:8, 0:20, PayloadLength:16, 17:8, 64:8, Source_address/binary,
329         Destination_address/binary, UdpPckt/binary, Payload/binary>>,
330
331     Tf = 2#11,
332     Nh = 1,
333     Hlim = 2#10,
334     Cid = 0,
335     Sac = 0,
336     Sam = 2#01,
337     M = 0,
338     Dac = 0,
339     Dam = 2#01,
340     C = 0,
341     P = 2#01,
342     ExpectedCarriedInline = #{"SAM" => <<?Node1MacAddress/binary>>, "DAM" => <<?
343     Node2MacAddress/binary>>},
344
345     InlineData = <<?Node1MacAddress/binary, ?Node2MacAddress/binary>>,
346     UdpInline = <<1025:16, 177:8, 63628:16>>,
347
348     io:format("UdpInline ~p~n", [UdpInline]),
349     ExpectedHeader =
350         <<?IPHC_DHTYPE:3, Tf:2, Nh:1, Hlim:2, Cid:1, Sac:1, Sam:2, M:1, Dac:1, Dam
351         :2, InlineData/binary, ?UDP_DHTYPE:5, C:1, P:2, UdpInline/binary>>,
352
353     Pckt = <<Ipv6Pckt/binary, UdpPckt/binary>>,
354     {CompressedHeader, CarriedInlineData} = lowpan_core:compressIpv6Header(Pckt,
355         false),
356
357     io:format("Expected ~p~nReceived ~p~n", [ExpectedHeader, CompressedHeader]),
358     ExpectedHeader = CompressedHeader,
359
360     ExpectedCarriedInline = CarriedInlineData,

```

```

354     ok.
355
356 tcp_nh_pckt_comp(_Config) ->
357     Payload = <<"Testing basic IPHC compression with link-local address">>,
358     IPv6Header =
359         #ipv6_header{
360             version = 6,
361             traffic_class = 0,
362             flow_label = 0,
363             payload_length = byte_size(Payload),
364             % TCP
365             next_header = 6,
366             hop_limit = 64,
367             source_address = <<16#FE80:16, 0:48, ?Node1MacAddress/binary>>,
368             destination_address = <<16#FE80:16, 0:48, ?Node2MacAddress/binary>>
369         },
370
371     Ipv6Pckt = ipv6:buildIpv6Packet(IPv6Header, Payload),
372
373     Tf = 2#11,
374     Nh = 0,
375     Hlim = 2#10,
376     Cid = 0,
377     Sac = 0,
378     Sam = 2#01,
379     M = 0,
380     Dac = 0,
381     Dam = 2#01,
382     ExpectedCarriedInline =
383         #{
384             "SAM" => <<?Node1MacAddress/binary>>,
385             "DAM" => <<?Node2MacAddress/binary>>,
386             "NextHeader" => 6
387         },
388
389     InlineData = <<6:8, ?Node1MacAddress/binary, ?Node2MacAddress/binary>>,
390     ExpectedHeader =
391         <<?IPHC_DHTYPE:3, Tf:2, Nh:1, Hlim:2, Cid:1, Sac:1, Sam:2, M:1, Dac:1, Dam
392             :2, InlineData/binary>>,
393
394     {CompressedHeader, CarriedInlineData} = lowpan_core:compressIpv6Header(
395         Ipv6Pckt, false),
396
397     io:format("Expected ~p~nReceived ~p~n", [ExpectedHeader, CompressedHeader]),
398     ExpectedHeader = CompressedHeader,
399
400     ExpectedCarriedInline = CarriedInlineData,
401     ok.
402
403 icmp_nh_pckt_comp(_Config) ->
404     Payload = <<"Testing basic IPHC compression with link-local address">>,
405     IPv6Header =
406         #ipv6_header{
407             version = 6,
408             traffic_class = 0,
409             flow_label = 0,
410             payload_length = byte_size(Payload),
411             %ICMPv6
412             next_header = 58,
413             hop_limit = 255,
414             source_address = <<16#FE80:16, 0:48, ?Node1MacAddress/binary>>,
415             destination_address = <<16#FE80:16, 0:48, ?Node2MacAddress/binary>>

```

```

414     },
415
416     Ipv6Pckt = ipv6:buildIpv6Packet(IPv6Header, Payload),
417
418     Tf = 2#11,
419     Nh = 0,
420     Hlim = 2#11,
421     Cid = 0,
422     Sac = 0,
423     Sam = 2#01,
424     M = 0,
425     Dac = 0,
426     Dam = 2#01,
427     ExpectedCarriedInline =
428     #{
429         "SAM" => <<?Node1MacAddress/binary>>,
430         "DAM" => <<?Node2MacAddress/binary>>,
431         "NextHeader" => 58
432     },
433
434     InlineData = <<58:8, ?Node1MacAddress/binary, ?Node2MacAddress/binary>>,
435     ExpectedHeader =
436     <<?IPHC_DHTYPE:3, Tf:2, Nh:1, Hlim:2, Cid:1, Sac:1, Sam:2, M:1, Dac:1, Dam
437     :2, InlineData/binary>>,
438
439     {CompressedHeader, CarriedInlineData} = lowpan_core:compressIpv6Header(
440     Ipv6Pckt, false),
441
442     io:format("Expected ~p~nReceived ~p~n", [ExpectedHeader, CompressedHeader]),
443     ExpectedHeader = CompressedHeader,
444
445     ExpectedCarriedInline = CarriedInlineData,
446     ok.
447
448 %---Online resource (https://www.youtube.com/watch?v=0JMV03HN0xo&t=778s)
449 compress_header_example1_test(_Config) ->
450 Payload = <<"Hello world this is an ipv6 packet">>,
451 PayloadLength = byte_size(Payload),
452
453 SrcAddress = <<16#FE80:16, 0:48, 16#020164FFFE2FFCOA:64>>,
454 DstAddress = <<16#FF02:16, 0:48, 16#0000000000000001:64>>,
455 Ipv6Pckt =
456 <<6:4, 224:8, 0:20, PayloadLength:16, 58:8, 255:8, SrcAddress/binary,
457 DstAddress/binary, Payload/bitstring>>,
458
459 Tf = 2#10,
460 Nh = 0,
461 Hlim = 2#11,
462 Cid = 0,
463 Sac = 0,
464 Sam = 2#11,
465 M = 1,
466 Dac = 0,
467 Dam = 2#11,
468 ExpectedCarriedInline =
469     #{
470         "DAM" => <<1>>,
471         "NextHeader" => 58,
472         "TrafficClass" => 224
473     },
474     InlineData = <<0:2, 56:6, 58:8, 1:8>>,
475     ExpectedHeader =

```

```

473     <<?IPHC_DHTYPE:3, Tf:2, Nh:1, Hlim:2, Cid:1, Sac:1, Sam:2, M:1, Dac:1, Dam
         :2, InlineData/binary>>,
474
475     {CompressedHeader, CarriedInlineData} = lowpan_core:compressIpv6Header(
         Ipv6Pckt, false),
476     io:format("Expected ~p~nReceived ~p~n", [ExpectedHeader, CompressedHeader]),
477
478     ExpectedHeader = CompressedHeader,
479
480     ExpectedCarriedInline = CarriedInlineData,
481     ok.
482
483 compress_header_example2_test(_Config) ->
484     Payload = <<"Hello world this is an ipv6 packet">>,
485     PayloadLength = byte_size(Payload),
486
487     SrcAddress = <<16#2001066073013728:64, 16#0223DFFFFEA9F7AC:64>>,
488     DstAddress = <<16#2001A45040070803:64, 16#0000000000001004:64>>,
489     Ipv6Pckt =
490         <<6:4, 0:8, 0:20, PayloadLength:16, 6:8, 64:8, SrcAddress/binary,
         DstAddress/binary, Payload/binary>>,
491
492     Tf = 2#11,
493     Nh = 0,
494     Hlim = 2#10,
495     Cid = 0,
496     Sac = 1,
497     Sam = 2#00,
498     M = 0,
499     Dac = 1,
500     Dam = 2#00,
501     InlineData = <<6:8, SrcAddress/binary, DstAddress/binary>>,
502     ExpectedHeader =
503         <<?IPHC_DHTYPE:3, Tf:2, Nh:1, Hlim:2, Cid:1, Sac:1, Sam:2, M:1, Dac:1, Dam
         :2, InlineData/binary>>,
504
505     {CompressedHeader, _} = lowpan_core:compressIpv6Header(Ipv6Pckt, false),
506     io:format("Expected ~p~nActual ~p~n", [ExpectedHeader, CompressedHeader]),
507
508     ExpectedHeader = CompressedHeader,
509
510     ok.
511
512 %-----
513 %               6LoWPAN IPv6 Packet Fragmentation
514 %-----
515
516 fragmentation_test(_Config) ->
517     % fragmentation test based on the computation of the size of all fragment
         payloads
518     Payload = <<"This is an Ipv6 pkt">>,
519     Ipv6Header =
520         #ipv6_header{
521             version = 6,
522             traffic_class = 0,
523             flow_label = 0,
524             payload_length = byte_size(Payload),
525             next_header = 17,
526             hop_limit = 64,
527             source_address = <<1>>,
528             destination_address = <<2>>
529         },

```

```

530     IPv6Pckt = ipv6:buildIPv6Packet(IPv6Header, Payload),
531     Fragments = lowpan_core:fragmentIPv6Packet(IPv6Pckt, byte_size(Payload), false
),
532     ReassembledSize =
533         lists:foldl(fun({_ , Fragment}, Acc) -> byte_size(Fragment) + Acc end, 0,
                    Fragments),
534     Psize = byte_size(IPv6Pckt),
535     Psize = ReassembledSize,
536     ok.
537
538 datagram_info_test(_Config) ->
539     Data = <<"payload">>,
540     Fragment = <<?FRAG1_DHTYPE:5, 1000:11, 12345:16, Data/bitstring>>,
541
542     DtgInfo = lowpan_core:datagramInfo(Fragment),
543     FragType = DtgInfo#datagramInfo.fragtype,
544     DatagramSize = DtgInfo#datagramInfo.datagramSize,
545     DatagramTag = DtgInfo#datagramInfo.datagramTag,
546     DatagramOffset = DtgInfo#datagramInfo.datagramOffset,
547     Payload = DtgInfo#datagramInfo.payload,
548
549     io:format("~p~n", [Payload]),
550
551     ?FRAG1_DHTYPE = FragType,
552     1000 = DatagramSize,
553     12345 = DatagramTag,
554     0 = DatagramOffset,
555     Data = Payload,
556     ok.
557
558 %-----
559 %                               Ipv6 Packet Reassembly
560 %-----
561
562 reassemble_fragments_list_test(_Config) ->
563     Data = <<"Hello World!">>,
564     PayloadLen = byte_size(Data),
565     Datagram = #datagram{
566         tag = 25,
567         size = PayloadLen,
568         cmpt = PayloadLen,
569         fragments = #{0 => <<"Hello ">>, 1 => <<"World!">>},
570         timer = erlang:system_time(second)
571     },
572
573     Reassembled = lowpan_core:reassemble(Datagram),
574     <<"Hello World!">> = Reassembled,
575     ok.
576
577 reassemble_single_fragments_test(_Config) ->
578     Data = <<"Hello World!">>,
579     PayloadLen = byte_size(Data),
580
581     DatagramMap = ets:new(datagram_map_test, [named_table, public]),
582     {Result1, _Map1} = lowpan_core:storeFragment(DatagramMap, {<<1>>, 25}, 0, <<"
                    Hello ">>, erlang:system_time(second), PayloadLen, 25, self()),
583     incomplete_first = Result1,
584
585     {Result2, _Map2} = lowpan_core:storeFragment(DatagramMap, {<<1>>, 25}, 1, <<"
                    World!">>, erlang:system_time(second), PayloadLen, 25, self()),
586     complete = Result2,
587

```

```

588     Reassembled = lowpan_core:reassemble(#datagram{
589         tag = 25,
590         size = PayloadLen,
591         cmpt = PayloadLen,
592         timer = erlang:system_time(second),
593         fragments = #{0 => <<"Hello ">>, 1 => <<"
                    World!">>}
594     }),
595     Data = Reassembled,
596     ets:delete(DatagramMap),
597     ok.
598
599 reassemble_full_ipv6_pkt_test(_Config) ->
600     Payload = lowpan_core:generateChunks(),
601     IPv6Header =
602         #ipv6_header{
603             version = 6,
604             traffic_class = 0,
605             flow_label = 0,
606             payload_length = byte_size(Payload),
607             next_header = 17,
608             hop_limit = 64,
609             source_address = <<1:128>>,
610             destination_address = <<2:128>>
611         },
612
613     Ipv6Pckt = ipv6:buildIpv6Packet(IPv6Header, Payload),
614     io:format("Original pkt size ~p bytes~n", [byte_size(Ipv6Pckt)]),
615     FragmentList = lowpan_core:fragmentIpv6Packet(Ipv6Pckt, byte_size(Ipv6Pckt),
        false),
616
617     DatagramMap = ets:new(datagram_map_test, [named_table, public]),
618
619     lists:foreach(
620         fun({FragHeader, FragPayload}) ->
621             Offset = case byte_size(FragHeader) of
622                 4 -> %first frag
623                     0;
624                 5 ->
625                     <<_:32, Offset:8>> = FragHeader,
626                     Offset
627             end,
628
629             io:format("Storing fragment with offset ~p~n", [Offset]),
630             {Result, _Map} = lowpan_core:storeFragment(DatagramMap, {<<1>>, 25},
                Offset, FragPayload, erlang:system_time(second), byte_size(
                Ipv6Pckt), 25, self()),
631             io:format("Fragment stored result: ~p~n", [Result])
632         end,
633         FragmentList
634     ),
635
636     Datagram = ets:lookup_element(DatagramMap, {<<1>>, 25}, 2),
637     io:format("Datagram after storing fragments: ~p~n", [Datagram]),
638
639     Reassembled = lowpan_core:reassemble(Datagram),
640     io:format("Reassembled: ~p~nIpv6Pckt: ~p~n", [Reassembled, Ipv6Pckt]),
641
642     case Ipv6Pckt of
643         Reassembled -> io:format("Reassembly successful.~n");
644         _ -> io:format("Reassembly failed.~n")
645     end,

```

```

646 % Nettoyage
647 ets:delete(DatagramMap),
648 ok.
649
650
651
652 %-----
653 %                               Additionnal tests
654 %-----
655 extended_EUI64_from_48mac(_Config)->
656   MacAddr = <<16#9865FD361453:48>>,
657   Expected = <<16#9A65FDFFE361453:64>>,
658   Result = lowpan_core:getEUI64From48bitMac(MacAddr),
659   io:format("Expected ~p~nResult ~p~n",[Expected, Result]),
660   Result = Expected.
661
662 extended_EUI64_from_64mac(_Config)->
663   MacAddr = <<16#00124B0006386C1A:64>>,
664   Expected = <<16#02124B0006386C1A:64>>,
665   Result = lowpan_core:getEUI64FromExtendedMac(MacAddr),
666   io:format("Expected ~p~nResult ~p~n",[Expected, Result]),
667   Result = Expected.
668
669 extended_EUI64_from_16mac(_Config)->
670   MacAddr = <<16#0001:16>>,
671   Expected = <<16#FDFF:16, 0:8, 16#FFFE:16, 0:8, 16#0001:16>>,
672   Result = lowpan_core:getEUI64FromShortMac(MacAddr),
673   io:format("Expected ~p~nResult ~p~n",[Expected, Result]),
674   Result = Expected.
675
676 link_local_from_16mac(_Config)->
677   MacAddr = <<16#0001:16>>,
678   Expected = <<16#FE80:16, 0:48,16#FDFF:16, 0:8, 16#FFFE:16, 0:8, 16#0001:16>>,
679   Result = lowpan_core:generateLLAddr(MacAddr),
680   io:format("Expected ~p~nResult ~p~n",[Expected, Result]),
681   Result = Expected.
682
683 check_tag_unicity(_Config) ->
684   TagMap = #{},
685   Tag1 = 10,
686   {_, NewTagMap} = lowpan_core:checkTagUnicity(TagMap, Tag1),
687   Tag2 = 10,
688   {ValidTag2, FinalMap} = lowpan_core:checkTagUnicity(NewTagMap, Tag2),
689   io:format("TagMap: ~p~n", [FinalMap]),
690   ValidTag2 /= Tag2.
691
692
693 multicast_addr_validity(_Config) ->
694   Ipv6Addr = <<16#FF02:16, 0:64, 1:16, 16#FF00:16, 16#1234:16>>,
695   GenAddr = lowpan_core:generateMulticastAddr(Ipv6Addr),
696   ExpectedAddr = <<16#9234:16>>,
697   io:format("ExpectedAddr ~p~nGenAddr ~p~n", [ExpectedAddr, GenAddr]),
698   GenAddr = ExpectedAddr.
699
700
701 %
702 %                               Contiki-ng cooja packet
703 %
704 %-----

```

```

704 cooja_example1(_Config)->
705     Payload = <<"Cooja example 1">>,
706     PayloadLength = byte_size(Payload),
707
708     Source_address = <<16#FE80:16, 0:48, 207:16, 7:16, 7:16, 7:16>>,
709     Destination_address = <<16#FF02:16, 0:48, 0:48,16#1a:16>>,
710
711     Ipv6Pckt =
712         <<6:4, 0:8, 0:20, PayloadLength:16, 58:8, 64:8, Source_address/binary,
              Destination_address/binary, Payload/binary>>,
713
714     Tf = 2#11,
715     Nh = 0,
716     Hlim = 2#10,
717     Cid = 0,
718     Sac = 0,
719     Sam = 2#11,
720     M = 1,
721     Dac = 0,
722     Dam = 2#11,
723     <<_:120,Last8:8>> = Destination_address,
724     ExpectedCarriedInline = #{"NextHeader" => 58, "DAM"=><<Last8:8>>},
725
726     InlineData = <<58:8, Last8:8>>,
727     UdpInline = <<1025:16, 177:8, 63628:16>>,
728
729     %io:format("Expected UdpInline ~p~n", [UdpInline]),
730     ExpectedHeader =
731         <<?IPHC_DHTYPE:3, Tf:2, Nh:1, Hlim:2, Cid:1, Sac:1, Sam:2, M:1, Dac:1, Dam
              :2, InlineData/binary>>,
732
733     CH = {Tf, Nh, Hlim, Cid, Sac, Sam, M, Dac, Dam, InlineData},
734     io:format("CH: ~p~n",[CH]),
735     io:format("Expected carried values: ~p~n", [ExpectedCarriedInline]),
736     Pckt = <<Ipv6Pckt/binary>>,
737     {CompressedHeader, CarriedInlineData} = lowpan_core:compressIpv6Header(Pckt,
              false),
738
739
740
741     %io:format("Expected ~p~nReceived ~p~n", [ExpectedHeader, CompressedHeader]),
742     ExpectedHeader = CompressedHeader,
743
744     ExpectedCarriedInline = CarriedInlineData,
745     ok.
746
747 cooja_example2(_Config)->
748     Payload = <<"Cooja example 2">>,
749     PayloadLength = byte_size(Payload),
750
751     Source_address = <<16#FE80:16, 0:48, 202:16, 2:16, 2:16, 2:16>>,
752     Destination_address = <<16#FE80:16, 0:48, 212:16, 7402:16, 2:16, 2:16>>,
753
754     UdpPckt = <<5683:16, 5683:16, 37:16, 16#8441:16>>,
755
756     Ipv6Pckt =
757         <<6:4, 0:8, 0:20, PayloadLength:16, 17:8, 64:8, Source_address/binary,
              Destination_address/binary, UdpPckt/binary, Payload/binary>>,
758
759     Tf = 2#11,
760     Nh = 1,

```



```

761 Hlim = 2#10,
762 Cid = 0,
763 Sac = 0,
764 Sam = 2#11,
765 M = 0,
766 Dac = 0,
767 Dam = 2#11,
768 C = 0,
769 P = 2#00,
770 ExpectedCarriedInline = #{},
771
772 %InlineData = <<?Node1MacAddress/binary, ?Node2MacAddress/binary>>,
773 UdpInline = <<5683:16, 5683:8, 8441:16>>,
774
775 %io:format("Expected UdpInline ~p~n", [UdpInline]),
776 ExpectedHeader =
777     <<?IPHC_DHTYPE:3, Tf:2, Nh:1, Hlim:2, Cid:1, Sac:1, Sam:2, M:1, Dac:1, Dam
778         :2, ?UDP_DHTYPE:5, C:1, P:2, UdpInline/binary>>,
779
780 CH = {Tf, Nh, Hlim, Cid, Sac, Sam, M, Dac, Dam, UdpInline},
781 io:format("Expected carried values: ~p~n", [ExpectedCarriedInline]),
782 Pckt = <<Ipv6Pckt/binary, UdpPckt/binary>>,
783 {CompressedHeader, CarriedInlineData} = lowpan_core:compressIpv6Header(Pckt,
784     false),
785
786 %io:format("Expected ~p~nReceived ~p~n", [ExpectedHeader, CompressedHeader]),
787 ExpectedHeader = CompressedHeader,
788
789 ExpectedCarriedInline = CarriedInlineData,
790 ok.
791
792 cooja_example3(_Config)->
793 Payload = <<"Cooja example 3">>,
794 PayloadLength = byte_size(Payload),
795
796 Source_address = <<0:64, 207:16, 7:16, 7:16, 7:16>>,
797 Destination_address = <<0:64, 203:16, 3:16, 3:16, 3:16>>,
798
799 Ipv6Pckt =
800     <<6:4, 0:8, 0:20, PayloadLength:16, 43:8, 63:8, Source_address/binary,
801         Destination_address/binary, Payload/binary>>,
802
803 Tf = 2#11,
804 Nh = 1,
805 Hlim = 2#00,
806 Cid = 1,
807 Sac = 1,
808 Sam = 2#01,
809 M = 0,
810 Dac = 1,
811 Dam = 2#11,
812
813 <<_:64, Last64S:64>> = Source_address,
814
815 ExpectedCarriedInline = #{"HopLimit"=>63, "NH"=>43, "SAM" => <<Last64S:64>>},
816
817 InlineData = <<?Node1MacAddress/binary, ?Node2MacAddress/binary>>,
818 UdpInline = <<1025:16, 177:8, 63628:16>>,
819
820 %io:format("Expected UdpInline ~p~n", [UdpInline]),
821 ExpectedHeader =

```

```

820     <<?IPHC_DHTYPE:3, Tf:2, Nh:1, Hlim:2, Cid:1, Sac:1, Sam:2, M:1, Dac:1, Dam
      :2, InlineData/binary>>,
821
822     CH = {Tf, Nh, Hlim, Cid, Sac, Sam, M, Dac, Dam, InlineData},
823     io:format("Expected carried values: ~p~n", [ExpectedCarriedInline]),
824     Pckt = <<Ipv6Pckt/binary>>,
825
826     {CompressedHeader, CarriedInlineData} = lowpan_core:compressIpv6Header(Pckt,
      false),
827
828     %io:format("Expected ~p~nReceived ~p~n", [ExpectedHeader, CompressedHeader]),
829     ExpectedHeader = CompressedHeader,
830
831     ExpectedCarriedInline = CarriedInlineData,
832     ok.

```

A.10 Simulation tests code

```

1 -module(lowpan_sender_receiver_SUITE).
2
3 -include_lib("common_test/include/ct.hrl").
4 -include("../src/utills.hrl").
5
6 -export([
7     all/0, groups/0, init_per_suite/1, end_per_suite/1, init_per_group/2,
8     end_per_group/2, init_per_testcase/2, end_per_testcase/2,
9     simple_pckt_sender/1, simple_pckt_receiver/1, big_payload_sender/1,
10    big_payload_receiver/1, multicast_sender/1, unspecified_dst_sender/1,
11    routing_req_sender/1, routing_req_receiver2/1,
12    routing_req_receiver3/1, big_pyld_routing_sender/1, big_pyld_routing_receiver2
13    /1,
14    big_pyld_routing_receiver3/1, discarded_sender/1, discarded_receiver/1,
15    no_hoplft_dst_reached_sender/1, no_hoplft_dst_reached_receiver/1,
16    unexpected_dtg_size_sender/1, same_tag_different_senders_sender/1,
17    same_tag_different_senders_receiver/1,
18    timeout_sender/1, timeout_receiver/1, duplicate_sender/1,
19    duplicate_receiver/1, multiple_hop_sender/1, multiple_hop_receiver2/1,
20    multiple_hop_receiver3/1, multiple_hop_receiver4/1,
21    nalp_sender/1, broadcast_sender/1, broadcast_receiver/1,
22    extended_hopsleft_sender/1, extended_hopsleft_receiver2/1,
23    extended_hopsleft_receiver3/1, extended_hopsleft_receiver4/1,
24    mesh_prefix_sender/1, mesh_prefix_receiver/1,
25    simple_udp_pckt_sender/1, simple_udp_pckt_receiver/1
26 ]).
27
28 all() ->
29     [{group, test_scenarios}].
30
31 %----- Tests groups
32 -----
33
34 groups() ->
35     [
36         {test_scenarios, [], [
37             {group, simple_tx_rx},
38             {group, big_payload_tx_rx},
39             {group, multicast_src_tx},

```

```

34     {group, unspecified_dst_tx},
35     {group, routing_req_tx_rx},
36     {group, discard_datagram_tx_rx},
37     {group, no_hoplft_dst_reached_tx_rx},
38     {group, unexpected_dtg_size_tx},
39     {group, same_tag_different_senders},
40     {group, timeout_scenario},
41     {group, duplicate_tx_rx},
42     {group, multiple_hop_tx_rx},
43     {group, nalp_tx_rx},
44     {group, broadcast_tx_rx},
45     {group, extendedHopsleftTx_rx},
46     {group, big_pyld_routing_tx_rx},
47     {group, simple_udp_tx_rx},
48     {group, mesh_prefix_tx_rx}
49  ]},
50  {simple_tx_rx, [parallel, {repeat, 1}], [simple_pckt_sender,
51    simple_pckt_receiver]},
52  {simple_udp_tx_rx, [parallel, {repeat, 1}], [simple_udp_pckt_sender,
53    simple_udp_pckt_receiver]},
54  {big_payload_tx_rx, [parallel, {repeat, 1}], [big_payload_sender,
55    big_payload_receiver]},
56  {multicast_src_tx, [sequential], [multicast_sender]},
57  {unspecified_dst_tx, [sequential], [unspecified_dst_sender]},
58  {routing_req_tx_rx, [parallel, {repeat, 1}], [routing_req_sender,
59    routing_req_receiver3, routing_req_receiver2]},
60  {big_pyld_routing_tx_rx, [parallel, {repeat, 1}], [big_pyld_routing_sender
61    , big_pyld_routing_receiver2, big_pyld_routing_receiver3]},
62  {discard_datagram_tx_rx, [parallel, {repeat, 1}], [discarded_sender,
63    discarded_receiver]},
64  {no_hoplft_dst_reached_tx_rx, [parallel, {repeat, 1}], [
65    no_hoplft_dst_reached_sender, no_hoplft_dst_reached_receiver]},
66  {unexpected_dtg_size_tx, [sequential], [unexpected_dtg_size_sender]},
67  {same_tag_different_senders, [parallel, {repeat, 1}], [
68    same_tag_different_senders_sender, same_tag_different_senders_receiver
69    ]},
70  {timeout_scenario, [parallel, {repeat, 1}], [timeout_sender,
71    timeout_receiver]},
72  {duplicate_tx_rx, [parallel, {repeat, 1}], [duplicate_sender,
73    duplicate_receiver]},
74  {multiple_hop_tx_rx, [parallel, {repeat, 1}], [multiple_hop_sender,
75    multiple_hop_receiver2, multiple_hop_receiver3, multiple_hop_receiver4
76    ]},
77  {nalp_tx_rx, [sequential], [nalp_sender]},
78  {broadcast_tx_rx, [parallel, {repeat, 1}], [broadcast_sender,
79    broadcast_receiver]},
80  {extendedHopsleftTx_rx, [parallel, {repeat, 1}], [extended_hopsleft_sender
81    , extended_hopsleft_receiver2, extended_hopsleft_receiver3,
82    extended_hopsleft_receiver4]},
83  {mesh_prefix_tx_rx, [parallel, {repeat, 1}], [mesh_prefix_sender,
84    mesh_prefix_receiver]}
85  ].
86
87  %-----
88  init_per_group(simple_tx_rx, Config) ->
89    init_per_group_setup(?Node1Address, ?Node2Address, ?Payload, Config);
90  %-----
91  init_per_group(simple_udp_tx_rx, Config) ->
92    init_per_group_udp_setup(?Node1Address, ?Node2Address, <<"Hello world">>,
93    Config);
94  %-----
95  init_per_group(big_payload_tx_rx, Config) ->

```

```

78     init_per_group_setup(?Node1Address, ?Node2Address, ?BigPayload, Config);
79     %-----
80 init_per_group(multicast_src_tx, Config) ->
81     init_per_group_setup(<<16#FF:16, 0:112>>, ?Node2Address, ?Payload, Config);
82     %-----
83 init_per_group(unspecified_dst_tx, Config) ->
84     init_per_group_setup(?Node1Address, <<0:128>>, ?Payload, Config);
85     %-----
86 init_per_group(routing_req_tx_rx, Config) ->
87     init_per_group_setup(?Node1Address, ?Node2Address, ?Payload, Config);
88     %-----
89 init_per_group(big_pyld_routing_tx_rx, Config) ->
90     init_per_group_setup(?Node1Address, ?Node3Address, ?BigPayload, Config);
91     %-----
92 init_per_group(discard_datagram_tx_rx, Config) ->
93     init_per_group_setup(?Node1Address, ?Node2Address, ?Payload, Config);
94     %-----
95 init_per_group(no_hoplft_dst_reached_tx_rx, Config) ->
96     init_per_group_setup(?Node1Address, ?Node2Address, ?Payload, Config);
97     %-----
98 init_per_group(unexpected_dtg_size_tx, Config) ->
99     Payload = lowpan_core:generateChunks(120),
100    init_per_group_setup(?Node1Address, ?Node2Address, Payload, Config);
101    %-----
102 init_per_group(same_tag_different_senders, Config) ->
103    init_per_group_setup(?Node1Address, ?Node2Address, ?Payload, Config);
104    %-----
105 init_per_group(timeout_scenario, Config) ->
106    init_per_group_setup(?Node1Address, ?Node2Address, ?Payload, Config);
107    %-----
108 init_per_group(tag_verification_tx_rx, Config) ->
109    init_per_group_setup(?Node1Address, ?Node2Address, ?BigPayload, Config);
110    %-----
111 init_per_group(duplicate_tx_rx, Config) ->
112    init_per_group_setup(?Node1Address, ?Node2Address, ?Payload, Config);
113    %-----
114 init_per_group(multiple_hop_tx_rx, Config) ->
115    init_per_group_setup(?Node1Address, ?Node4Address, ?Payload, Config);
116    %-----
117 init_per_group(nalp_tx_rx, Config) ->
118    init_per_group_setup(?Node1Address, ?Node2Address, ?Payload, Config);
119    %-----
120 init_per_group(broadcast_tx_rx, Config) ->
121    init_per_group_setup(?Node1Address, <<16#FF02:16, 0:64, 1:16, 16#FF00:16,
122    16#1234:16>>, ?Payload, Config);
123    %-----
124 init_per_group(extendedHopsleftTx_rx, Config) ->
125    init_per_group_setup(?Node1Address, ?Node4Address, ?Payload, Config);
126    %-----
127 init_per_group(mesh_prefix_tx_rx, Config) ->
128    MacAddress = lowpan_core:generateEUI64MacAddr(?Node2MacAddress),
129    init_per_group_setup(?Node1Address, <<?MESH_LOCAL_PREFIX:16, 16#0DB8:16, 0:32,
130    MacAddress/binary>>, ?Payload, Config);
131    %-----
132 init_per_group(benchmark_tx_rx, Config) ->
133    init_per_group_setup(?Node1Address, ?Node2Address, ?Payload, Config);
134    %-----
135 init_per_group(_, Config) ->
136    Config.
137
138 init_per_group_setup(Src, Dst, Payload, Config) ->
139    {NetPid, Network} = lowpan_node:boot_network_node(#{loss => true}),

```

```

138     io:format("Initializing group ~n"),
139
140     IPv6Header = #ipv6_header{
141         version = 6,
142         traffic_class = 0,
143         flow_label = 0,
144         payload_length = byte_size(Payload),
145         next_header = 12,
146         hop_limit = 64,
147         source_address = Src,
148         destination_address = Dst
149     },
150     Packet = ipv6:buildIpv6Packet(IPv6Header, Payload),
151     [
152         {net_pid, NetPid},
153         {network, Network},
154         {node1_mac_address, ?Node1MacAddress},
155         {node2_mac_address, ?Node2MacAddress},
156         {node3_mac_address, ?Node3MacAddress},
157         {node4_mac_address, ?Node4MacAddress},
158         {ipv6_packet, Packet}
159     | Config
160     ].
161
162 init_per_group_udp_setup(Src, Dst, Payload, Config) ->
163     [{NetPid, Network} = lowpan_node:boot_network_node(#{loss => true}),
164     io:format("Initializing group ~n"),
165
166     PayloadLength = byte_size(Payload),
167     IPv6Header =
168         #ipv6_header{
169             version = 6,
170             traffic_class = 0,
171             flow_label = 0,
172             % 4 bytes for the UDP header
173             payload_length = PayloadLength,
174             next_header = 17,
175             hop_limit = 64,
176             source_address = Src,
177             destination_address = Dst
178         },
179     UdpHeader =
180         #udp_header{
181             source_port = 1025,
182             destination_port = 61617,
183             length = PayloadLength,
184             checksum = 16#f88c
185         },
186
187     Packet = ipv6:buildIpv6UdpPacket(IPv6Header, UdpHeader, Payload),
188     [
189         {net_pid, NetPid},
190         {network, Network},
191         {node1_mac_address, ?Node1MacAddress},
192         {node2_mac_address, ?Node2MacAddress},
193         {node3_mac_address, ?Node3MacAddress},
194         {node4_mac_address, ?Node4MacAddress},
195         {ipv6_packet, Packet}
196     | Config
197     ].
198
199 end_per_group(_Group, Config) ->

```

```

200     Network = proplists:get_value(network, Config),
201     NetPid = proplists:get_value(net_pid, Config),
202
203     if
204         Network == undefined ->
205             io:format("Error: Network not found in Config-n"),
206             {error, network_not_found};
207         NetPid == undefined ->
208             io:format("Error: NetPid not found in Config-n"),
209             {error, net_pid_not_found};
210         true ->
211             lowpan_node:stop_network_node(Network, NetPid),
212             ok
213     end.
214
215
216
217 %----- Tests cases initialization
218 -----
219 default_sender_init_per_testcase(Config, RoutingTable)->
220     Network = ?config(network, Config),
221     Node1MacAddress = ?config(node1_mac_address, Config),
222     Node = lowpan_node:boot_lowpan_node(node1, Network, Node1MacAddress,
223         RoutingTable),
224     [{node1, Node} | Config].
225
226 default_receiver2_init_per_testcase(Config, RoutingTable)->
227     Network = ?config(network, Config),
228     Node2MacAddress = ?config(node2_mac_address, Config),
229     Callback = fun lowpan_api:inputCallback/4,
230     Node = lowpan_node:boot_lowpan_node(node2, Network, Node2MacAddress, Callback,
231         RoutingTable),
232     [{node2, Node} | Config].
233
234 default_receiver3_init_per_testcase(Config, RoutingTable)->
235     Network = ?config(network, Config),
236     Node3MacAddress = ?config(node3_mac_address, Config),
237     Callback = fun lowpan_api:inputCallback/4,
238     Node = lowpan_node:boot_lowpan_node(node3, Network, Node3MacAddress, Callback,
239         RoutingTable),
240     [{node3, Node} | Config].
241
242 default_receiver4_init_per_testcase(Config, RoutingTable)->
243     Network = ?config(network, Config),
244     Node4MacAddress = ?config(node4_mac_address, Config),
245     Callback = fun lowpan_api:inputCallback/4,
246     Node = lowpan_node:boot_lowpan_node(node4, Network, Node4MacAddress, Callback,
247         RoutingTable),
248     [{node4, Node} | Config].
249
250 broadcast_receiver_init_per_testcase(Config, RoutingTable)->
251     Network = ?config(network, Config),
252     MacAddress = <<16#9234:16>>,
253     Callback = fun lowpan_api:inputCallback/4,
254     Node = lowpan_node:boot_lowpan_node(broadcast_node, Network, MacAddress,
255         Callback, RoutingTable),
256     [{broadcast_node, Node} | Config].
257
258 %-----
259 init_per_testcase(simple_pkt_sender, Config)->

```

```

256     default_sender_init_per_testcase(Config, ?Default_routing_table);
257
258 init_per_testcase(simple_pkt_receiver, Config)->
259     default_receiver2_init_per_testcase(Config, ?Default_routing_table);
260
261 %-----
262 init_per_testcase(simple_udp_pkt_sender, Config)->
263     default_sender_init_per_testcase(Config, ?Default_routing_table);
264
265 init_per_testcase(simple_udp_pkt_receiver, Config)->
266     default_receiver2_init_per_testcase(Config, ?Default_routing_table);
267
268 %-----
269 init_per_testcase(big_payload_sender, Config)->
270     default_sender_init_per_testcase(Config, ?Default_routing_table);
271
272 init_per_testcase(big_payload_receiver, Config)->
273     default_receiver2_init_per_testcase(Config, ?Default_routing_table);
274
275 %-----
276 init_per_testcase(multicast_sender, Config)->
277     Config1 = default_sender_init_per_testcase(Config, ?Default_routing_table),
278     default_receiver2_init_per_testcase(Config1, ?Default_routing_table);
279
280 %-----
281 init_per_testcase(unspecified_dst_sender, Config)->
282     Config1 = default_sender_init_per_testcase(Config, ?Default_routing_table),
283     default_receiver2_init_per_testcase(Config1, ?Default_routing_table);
284
285 %-----
286 init_per_testcase(routing_req_sender, Config)->
287     default_sender_init_per_testcase(Config, ?Node1_routing_table);
288
289 init_per_testcase(routing_req_receiver2, Config)->
290     default_receiver2_init_per_testcase(Config, ?Node2_routing_table);
291
292 init_per_testcase(routing_req_receiver3, Config)->
293     default_receiver3_init_per_testcase(Config, ?Node3_routing_table);
294
295 %-----
296 init_per_testcase(big_pyld_routing_sender, Config)->
297     default_sender_init_per_testcase(Config, ?Node1_routing_table);
298
299 init_per_testcase(big_pyld_routing_receiver2, Config)->
300     default_receiver2_init_per_testcase(Config, ?Node2_routing_table);
301
302 init_per_testcase(big_pyld_routing_receiver3, Config)->
303     default_receiver3_init_per_testcase(Config, ?Node3_routing_table);
304
305 %-----
306 init_per_testcase(discarded_sender, Config)->
307     default_sender_init_per_testcase(Config, ?Node1_routing_table);
308
309 init_per_testcase(discarded_receiver, Config)->
310     default_receiver2_init_per_testcase(Config, ?Node2_routing_table);
311
312 %-----
313 init_per_testcase(no_hoplft_dst_reached_sender, Config)->
314     default_sender_init_per_testcase(Config, ?Node1_routing_table);
315
316 init_per_testcase(no_hoplft_dst_reached_receiver, Config)->
317     default_receiver2_init_per_testcase(Config, ?Node2_routing_table);

```

```

318
319 %-----
320 init_per_testcase(unexpected_dtg_size_sender, Config)->
321     default_sender_init_per_testcase(Config, ?Node1_routing_table);
322
323 %-----
324 init_per_testcase(same_tag_different_senders_sender, Config) ->
325     Config1 = default_sender_init_per_testcase(Config, ?Default_routing_table),
326     default_receiver2_init_per_testcase(Config1, ?Default_routing_table);
327
328 init_per_testcase(same_tag_different_senders_receiver, Config) ->
329     default_receiver3_init_per_testcase(Config, ?Default_routing_table);
330
331 %-----
332 init_per_testcase(timeout_sender, Config) ->
333     default_sender_init_per_testcase(Config, ?Default_routing_table);
334 init_per_testcase(timeout_receiver, Config) ->
335     default_receiver2_init_per_testcase(Config, ?Default_routing_table);
336
337 %-----
338 init_per_testcase(tag_verification_sender, Config) ->
339     default_sender_init_per_testcase(Config, ?Default_routing_table);
340
341 init_per_testcase(tag_verification_receiver, Config) ->
342     default_receiver2_init_per_testcase(Config, ?Default_routing_table);
343
344 %-----
345 init_per_testcase(duplicate_sender, Config) ->
346     default_sender_init_per_testcase(Config, ?Default_routing_table);
347
348 init_per_testcase(duplicate_receiver, Config) ->
349     default_receiver2_init_per_testcase(Config, ?Default_routing_table);
350
351 %-----
352 init_per_testcase(multiple_hop_sender, Config) ->
353     default_sender_init_per_testcase(Config, ?Node1_multiple_hop_routing_table);
354
355 init_per_testcase(multiple_hop_receiver2, Config)->
356     default_receiver2_init_per_testcase(Config, ?Node2_multiple_hop_routing_table);
357
358 init_per_testcase(multiple_hop_receiver3, Config)->
359     default_receiver3_init_per_testcase(Config, ?Node3_multiple_hop_routing_table);
360
361 init_per_testcase(multiple_hop_receiver4, Config) ->
362     default_receiver4_init_per_testcase(Config, ?Node4_multiple_hop_routing_table);
363
364 %-----
365 init_per_testcase(nalp_sender, Config) ->
366     default_sender_init_per_testcase(Config, ?Default_routing_table);
367
368 %-----
369 init_per_testcase(broadcast_sender, Config) ->
370     default_sender_init_per_testcase(Config, ?Default_routing_table);
371
372 init_per_testcase(broadcast_receiver, Config) ->
373     broadcast_receiver_init_per_testcase(Config, ?Default_routing_table);
374 %-----
375 init_per_testcase(extended_hopsleft_sender, Config) ->
376     default_sender_init_per_testcase(Config, ?Node1_multiple_hop_routing_table);
377
378 init_per_testcase(extended_hopsleft_receiver2, Config)->
379     default_receiver2_init_per_testcase(Config, ?Node2_multiple_hop_routing_table);

```



```

380
381 init_per_testcase(extended_hopsleft_receiver3, Config)->
382     default_receiver3_init_per_testcase(Config, ?Node3_multiple_hop_routing_table);
383
384 init_per_testcase(extended_hopsleft_receiver4, Config) ->
385     default_receiver4_init_per_testcase(Config, ?Node4_multiple_hop_routing_table);
386
387 %-----
388 init_per_testcase(mesh_prefix_sender, Config) ->
389     default_sender_init_per_testcase(Config, ?Default_routing_table);
390
391 init_per_testcase(mesh_prefix_receiver, Config) ->
392     Network = ?config(network, Config),
393     Callback = fun lowpan_api:inputCallback/4,
394     Node = lowpan_node:boot_lowpan_node(node2, Network, ?Node2MacAddress, Callback
395         , ?Default_routing_table),
396     [{node2, Node} | Config];
397 %-----
398 init_per_testcase(benchmark_sender, Config)->
399     default_sender_init_per_testcase(Config, ?Default_routing_table);
400
401 init_per_testcase(benchmark_receiver, Config)->
402     default_receiver2_init_per_testcase(Config, ?Default_routing_table);
403 %-----
404 init_per_testcase(_, Config) ->
405     stop_node(?config(node1, Config)),
406     stop_node(?config(node2, Config)),
407     stop_node(?config(node3, Config)),
408     stop_node(?config(node4, Config)),
409     Config.
410 stop_node({Pid, Node}) ->
411     case is_node_alive(Node) of
412     true ->
413         case catch erpc:call(Node, lowpan_node, stop_lowpan_node, [Node, Pid])
414             of
415             ok -> ok;
416             {'EXIT', Reason} ->
417                 io:format("Error stopping node ~p: ~p~n", [Node, Reason]),
418                 {error, stopping_node_failed}
419             end;
420     false ->
421         io:format("Node ~p is already stopped or not reachable.~n", [Node]),
422         ok
423     end;
424 stop_node(undefined) ->
425     io:format("Node was not started.~n"),
426     ok.
427 is_node_alive(Node) ->
428     case catch erpc:call(Node, some_module, ping, []) of
429     pong -> true;
430     _ -> false
431     end.
432
433
434 end_per_testcase(_, _) ->
435     ok.
436
437 init_per_suite(Config) ->
438     Config.
439

```

```

440 end_per_suite(_Config) ->
441     ok.
442
443 %-----
444 % Send a single payload from node 1 to node 2
445 %-----
446 simple_pkt_sender(Config) ->
447     {Pid1, Node1} = ?config(node1, Config),
448     IPv6Pckt = ?config(ipv6_packet, Config),
449     ok = erpc:call(Node1, lowpan_api, sendPacket, [IPv6Pckt, false]),
450     ct:pal("Payload sent successfully from node1 to node2"),
451     lowpan_node:stop_lowpan_node(Node1, Pid1).
452
453 %-----
454 % Payload from node 1 received by node 2
455 %-----
456 simple_pkt_receiver(Config) ->
457     {Pid2, Node2} = ?config(node2, Config),
458     IPv6Pckt = ?config(ipv6_packet, Config),
459
460     {CompressedHeader, _} = lowpan_core:compressIpv6Header(IPv6Pckt, false),
461     PcktInfo = lowpan_core:getPcktInfo(IPv6Pckt),
462     Payload = PcktInfo#ipv6PckInfo.payload,
463     CompressedIpv6Packet = <<CompressedHeader/binary, Payload/bitstring>>,
464
465     ReceivedData = erpc:call(Node2, lowpan_api, frameReception, []),
466
467     io:format("Expected: ~p~n~nReceived: ~p~n", [CompressedIpv6Packet,
468         ReceivedData]),
469     ReceivedData = CompressedIpv6Packet,
470
471     ct:pal("Payload received successfully at node2"),
472     lowpan_node:stop_lowpan_node(Node2, Pid2).
473
474 %-----
475 % Send a packet with udp as next header from node 1 to node 2
476 %-----
477 simple_udp_pkt_sender(Config) ->
478     {Pid1, Node1} = ?config(node1, Config),
479     Ipv6Pckt = ?config(ipv6_packet, Config),
480     ok = erpc:call(Node1, lowpan_api, sendPacket, [Ipv6Pckt, false]),
481     ct:pal("Payload sent successfully from node1 to node2"),
482     lowpan_node:stop_lowpan_node(Node1, Pid1).
483
484 %-----
485 % udp pkt reception from node 1 to node 2
486 %-----
487 simple_udp_pkt_receiver(Config) ->
488     {Pid2, Node2} = ?config(node2, Config),
489     IPv6Pckt = ?config(ipv6_packet, Config),
490
491     {CompressedHeader, _} = lowpan_core:compressIpv6Header(IPv6Pckt, false),
492     PcktInfo = lowpan_core:getPcktInfo(IPv6Pckt),
493     Payload = PcktInfo#ipv6PckInfo.payload,
494     CompressedIpv6Packet = <<CompressedHeader/binary, Payload/bitstring>>,
495
496     ReceivedData = erpc:call(Node2, lowpan_api, frameReception, []),
497
498     %io:format("Expected: ~p~n~nReceived: ~p~n", [CompressedIpv6Packet,
499         ReceivedData]),
499     ReceivedData = CompressedIpv6Packet,

```

```

500
501     ct:pal("Payload received successfully at node2"),
502     lowpan_node:stop_lowpan_node(Node2, Pid2).
503
504 %-----
505 % Send a large payload from node 1 to node 3
506 %-----
507 big_payload_sender(Config) ->
508     {Pid1, Node1} = ?config(node1, Config),
509     IPv6Pckt2 = ?config(ipv6_packet, Config),
510     io:format("Size ~p~n",[byte_size(IPv6Pckt2)]),
511     ok = erpc:call(Node1, lowpan_api, sendPacket, [IPv6Pckt2, false]),
512     ct:pal("Big payload sent successfully from node1 to node3"),
513     lowpan_node:stop_lowpan_node(Node1, Pid1).
514
515 %-----
516 % Large payload from node 1 received by node 3
517 %-----
518 big_payload_receiver(Config) ->
519     {Pid2, Node2} = ?config(node2, Config),
520     IPv6Pckt = ?config(ipv6_packet, Config),
521
522     {CompressedHeader, _} = lowpan_core:compressIpv6Header(IPv6Pckt, false),
523     PcktInfo = lowpan_core:getPcktInfo(IPv6Pckt),
524     Payload = PcktInfo#ipv6PcktInfo.payload,
525     CompressedIpv6Packet = <<CompressedHeader/binary, Payload/bitstring>>,
526
527     ReceivedData = erpc:call(Node2, lowpan_api, frameReception, []),
528
529     %io:format("Expected: ~p~n~nReceived: ~p~n", [CompressedIpv6Packet,
530         ReceivedData]),
531     ReceivedData = CompressedIpv6Packet,
532
533     ct:pal("Big payload received successfully at node2"),
534     lowpan_node:stop_lowpan_node(Node2, Pid2).
535
536 %-----
537 % Send packet with a multicast source address
538 %-----
539 multicast_sender(Config) ->
540     {Pid1, Node1} = ?config(node1, Config),
541     {Pid2, Node2} = ?config(node2, Config),
542
543     IPv6Pckt = ?config(ipv6_packet, Config),
544     error_multicast_src = erpc:call(Node1, lowpan_api, sendPacket, [IPv6Pckt,
545         false]),
546     ct:pal("Multicast Source address done"),
547     lowpan_node:stop_lowpan_node(Node1, Pid1),
548     lowpan_node:stop_lowpan_node(Node2, Pid2).
549
550 %-----
551 % Send packet with the unspecified dest address
552 %-----
553 unspecified_dst_sender(Config) ->
554     {Pid1, Node1} = ?config(node1, Config),
555     {Pid2, Node2} = ?config(node2, Config),
556
557     IPv6Pckt = ?config(ipv6_packet, Config),
558     error_unspecified_addr = erpc:call(Node1, lowpan_api, sendPacket, [IPv6Pckt,
559         false]),
560     ct:pal("Unspecified Source address done"),

```

```

559     lowpan_node:stop_lowpan_node(Node1, Pid1),
560     lowpan_node:stop_lowpan_node(Node2, Pid2).
561
562
563 %-----
564 % Send a packet that needs routing from node 1 to node 2
565 %-----
566 routing_req_sender(Config) ->
567     {Pid1, Node1} = ?config(node1, Config),
568     IPv6Pckt = ?config(ipv6_packet, Config),
569     erpc:call(Node1, lowpan_api, sendPacket, [IPv6Pckt, false]),
570     ct:pal("Routed packet sent successfully from node1 to node2"),
571     lowpan_node:stop_lowpan_node(Node1, Pid1).
572
573 %-----
574 % Reception of a routed packet
575 %-----
576 routing_req_receiver2(Config) ->
577     {Pid2, Node2} = ?config(node2, Config),
578     IPv6Pckt = ?config(ipv6_packet, Config),
579
580     {CompressedHeader, _} = lowpan_core:compressIpv6Header(IPv6Pckt, false),
581     PcktInfo = lowpan_core:getPcktInfo(IPv6Pckt),
582     Payload = PcktInfo#ipv6PckInfo.payload,
583     CompressedIpv6Packet = <<CompressedHeader/binary, Payload/bitstring>>,
584
585     ReceivedData = erpc:call(Node2, lowpan_api, frameReception, []),
586
587     %io:format("Expected: ~p~n~nReceived: ~p~n", [CompressedIpv6Packet,
588         ReceivedData]),
589     ReceivedData = CompressedIpv6Packet,
590
591     ct:pal("Routed packet received successfully at node2"),
592     lowpan_node:stop_lowpan_node(Node2, Pid2).
593 routing_req_receiver3(Config) ->
594     {Pid3, Node3} = ?config(node3, Config),
595     erpc:call(Node3, lowpan_api, frameReception, []),
596     lowpan_node:stop_lowpan_node(Node3, Pid3).
597
598 %-----
599 % Send a big packet that needs routing from node 1 to node 3
600 %-----
601 big_pyld_routing_sender(Config) ->
602     {Pid1, Node1} = ?config(node1, Config),
603     IPv6Pckt = ?config(ipv6_packet, Config),
604     erpc:call(Node1, lowpan_api, sendPacket, [IPv6Pckt, false]),
605     ct:pal("Big routed packet sent successfully from node1 to node3"),
606     lowpan_node:stop_lowpan_node(Node1, Pid1).
607
608 %-----
609 % Reception of a big payload with routing by node 3
610 %-----
611
612 big_pyld_routing_receiver2(Config) ->
613     {Pid2, Node2} = ?config(node2, Config),
614     erpc:call(Node2, lowpan_api, frameReception, []),
615     lowpan_node:stop_lowpan_node(Node2, Pid2).
616
617 big_pyld_routing_receiver3(Config) ->
618     {Pid3, Node3} = ?config(node3, Config),
619

```

```

620 IPv6Pckt = ?config(ipv6_packet, Config),
621 {CompressedHeader, _} = lowpan_core:compressIPv6Header(IPv6Pckt, false),
622 PcktInfo = lowpan_core:getPcktInfo(IPv6Pckt),
623 Payload = PcktInfo#ipv6PckInfo.payload,
624 CompressedIPv6Packet = <<CompressedHeader/binary, Payload/bitstring>>,
625
626 ReceivedData = erpc:call(Node3, lowpan_api, frameReception, []),
627
628 %io:format("Expected: ~p~n~nReceived: ~p~n", [CompressedIPv6Packet,
629 ReceivedData]),
630
631 ct:pal("Routed packet received successfully at node2"),
632
633 lowpan_node:stop_lowpan_node(Node3, Pid3).
634
635
636 %-----
637 % Send a datagram with 1 as value for hop left to node 2
638 %-----
639 discarded_sender(Config) ->
640     {Pid1, Node1} = ?config(node1, Config),
641     MeshHeader =
642         #mesh_header{
643             v_bit = 0,
644             f_bit = 0,
645             hops_left = 1,
646             originator_address = ?node1_addr,
647             final_destination_address = ?node3_addr
648         },
649
650     BinMeshHeader = lowpan_core:buildMeshHeader(MeshHeader),
651
652     Datagram = <<BinMeshHeader/binary, ?Payload/bitstring>>, % meshHeader + Data
653
654     FC = #frame_control{ack_req = ?ENABLED,
655                     frame_type = ?FTYPE_DATA,
656                     src_addr_mode = ?EXTENDED,
657                     dest_addr_mode = ?EXTENDED},
658     MacHdr = #mac_header{src_addr = ?node1_addr,
659                       dest_addr = ?node2_addr},
660
661     ok = erpc:call(Node1, lowpan_api, tx, [Datagram, FC, MacHdr]),
662     ct:pal("Packet with 1 hop left sent successfully from node1 to node3"),
663     lowpan_node:stop_lowpan_node(Node1, Pid1).
664
665 %-----
666 % Discard datagram received from node 1
667 %-----
668 discarded_receiver(Config) ->
669     {Pid2, Node2} = ?config(node2, Config),
670     dtg_discarded = erpc:call(Node2, lowpan_api, frameReception, []),
671     lowpan_node:stop_lowpan_node(Node2, Pid2).
672
673
674 %-----
675 % Send a datagram with 0 as value for hop left to node 2
676 %-----
677 no_hoplft_dst_reached_sender(Config) ->
678     {Pid1, Node1} = ?config(node1, Config),
679     MeshHeader =
680         #mesh_header{

```

```

681         v_bit = 0,
682         f_bit = 0,
683         hops_left = 0,
684         originator_address = ?node1_addr,
685         final_destination_address = ?node2_addr
686     },
687
688     BinMeshHeader = lowpan_core:buildMeshHeader(MeshHeader),
689
690     Datagram = <<BinMeshHeader/binary, ?IPV6_DHTYPE:8, ?Payload/bitstring>>, %
691         meshHeader + Data
692
693     FC = #frame_control{ack_req = ?ENABLED,
694         frame_type = ?FTYPE_DATA,
695         src_addr_mode = ?EXTENDED,
696         dest_addr_mode = ?EXTENDED},
697     MacHdr = #mac_header{src_addr = ?node1_addr,
698         dest_addr = ?node2_addr},
699
700     ok = erpc:call(Node1, lowpan_api, tx, [Datagram, FC, MacHdr]),
701     ct:pal("Packet with 0 hop left sent successfully from node1 to node2"),
702     lowpan_node:stop_lowpan_node(Node1, Pid1).
703
704 %-----
705 % Reception of datagram with 0 as value for hop left
706 %-----
707 no_hoplft_dst_reached_receiver(Config) ->
708     {Pid2, Node2} = ?config(node2, Config),
709     Response = erpc:call(Node2, lowpan_api, frameReception, []),
710     Response = ?Payload,
711     lowpan_node:stop_lowpan_node(Node2, Pid2).
712
713 %-----
714 % Check if error is return when datagram size is unexpected
715 %-----
716 unexpected_dtg_size_sender(Config) ->
717     {Pid1, Node1} = ?config(node1, Config),
718     IPv6Pckt = ?config(ipv6_packet, Config),
719     error_frag_size = erpc:call(Node1, lowpan_api, sendPacket, [IPv6Pckt, false]),
720     lowpan_node:stop_lowpan_node(Node1, Pid1).
721
722 %-----
723 % Send payloads from node 1 and node 2 to node 3 with the same tag
724 %-----
725 same_tag_different_senders_sender(Config) ->
726     {Pid1, Node1} = ?config(node1, Config),
727     {Pid2, Node2} = ?config(node2, Config),
728
729     Data1 = <<"Hello ">>,
730     Data2 = <<"World!">>,
731     PayloadLen = byte_size(Data1) + byte_size(Data2),
732
733     FragHeader1 = #frag_header{
734         frag_type = ?FRAG1_DHTYPE,
735         datagram_size = PayloadLen,
736         datagram_tag = 25,
737         datagram_offset = 0
738     },
739     FragHeader2 = #frag_header{
740         frag_type = ?FRAGN_DHTYPE,
741         datagram_size = PayloadLen,
742         datagram_tag = 25,

```

```

742     datagram_offset = 1
743 },
744
745 Frag1 = lowpan_core:buildDatagramPckt(FragHeader1, Data1),
746 Frag2 = lowpan_core:buildDatagramPckt(FragHeader2, Data2),
747
748 MeshHeader1 =
749     #mesh_header{
750         v_bit = 0,
751         f_bit = 0,
752         hops_left = 14,
753         originator_address = ?node1_addr,
754         final_destination_address = ?node3_addr
755     },
756
757 BinMeshHeader1 = lowpan_core:buildMeshHeader(MeshHeader1),
758
759 FC1 = #frame_control{ack_req = ?ENABLED,
760             frame_type = ?FTYPE_DATA,
761             src_addr_mode = ?EXTENDED,
762             dest_addr_mode = ?EXTENDED},
763 MH1 = #mac_header{src_addr = ?node1_addr,
764             dest_addr = ?node3_addr},
765
766 MeshHeader2 =
767     #mesh_header{
768         v_bit = 0,
769         f_bit = 0,
770         hops_left = 14,
771         originator_address = ?node2_addr,
772         final_destination_address = ?node3_addr
773     },
774
775 BinMeshHeader2 = lowpan_core:buildMeshHeader(MeshHeader2),
776 FC2 = #frame_control{ack_req = ?ENABLED,
777             frame_type = ?FTYPE_DATA,
778             src_addr_mode = ?EXTENDED,
779             dest_addr_mode = ?EXTENDED},
780 MH2 = #mac_header{src_addr = ?node2_addr,
781             dest_addr = ?node3_addr},
782
783 ok = erpc:call(Node1, lowpan_api, tx, [<<BinMeshHeader1/binary, Frag1/
784     bitstring>>, FC1, MH1]),
785 ok = erpc:call(Node2, lowpan_api, tx, [<<BinMeshHeader2/binary, Frag1/
786     bitstring>>, FC2, MH2]),
787
788 ok = erpc:call(Node1, lowpan_api, tx, [<<BinMeshHeader1/binary, Frag2/
789     bitstring>>, FC1, MH1]),
790 ok = erpc:call(Node2, lowpan_api, tx, [<<BinMeshHeader2/binary, Frag2/
791     bitstring>>, FC2, MH2]),
792
793 ct:pal("Fragments sent from node1 and node2 to node3 with the same tag"),
794 lowpan_node:stop_lowpan_node(Node1, Pid1),
795 lowpan_node:stop_lowpan_node(Node2, Pid2).
796
797 %-----
798 % Reception of payloads from node 1 and node 2 by node 3 with the same tag
799 %-----
800 same_tag_different_senders_receiver(Config) ->
801     {Pid3, Node3} = ?config(node3, Config),
802
803 % Receive and reassemble the fragments

```

```

800 ReceivedData1 = erpc:call(Node3, lowpan_api, frameReception, []),
801 ReceivedData2 = erpc:call(Node3, lowpan_api, frameReception, []),
802
803 ExpectedData = <<"Hello World!">>,
804 %io:format("Expected: ~p~n~nReceived 1: ~p~n~nReceived 2: ~p~n", [ExpectedData
      , ReceivedData1, ReceivedData2]),
805
806 case (ReceivedData1 == ExpectedData) andalso (ReceivedData2 == ExpectedData)
      of
807     true ->
808         ct:pal("Payloads received successfully at node3 with the same tag from
      different senders"),
809         lowpan_node:stop_lowpan_node(Node3, Pid3);
810     false ->
811         ct:fail("Payloads did not match expected data"),
812         lowpan_node:stop_lowpan_node(Node3, Pid3)
813 end.
814
815
816 -----
817 % Send incomplete payload from node 1 to node 2 to trigger a timeout
818 -----
819 timeout_sender(Config) ->
820     {Pid1, Node1} = ?config(node1, Config),
821
822     Data = <<"Hello World!">>,
823     PayloadLen = byte_size(Data),
824
825     FragHeader1 = #frag_header{
826         frag_type = ?FRAG1_DHTYPE,
827         datagram_size = PayloadLen,
828         datagram_tag = 25,
829         datagram_offset = 0
830     },
831
832     Frag1 = lowpan_core:buildDatagramPckt(FragHeader1, <<"Hello ">>),
833     MeshHeader1 =
834         #mesh_header{
835             v_bit = 0,
836             f_bit = 0,
837             hops_left = 14,
838             originator_address = ?node1_addr,
839             final_destination_address = ?node2_addr
840         },
841
842     BinMeshHeader1 = lowpan_core:buildMeshHeader(MeshHeader1),
843
844     FC1 = #frame_control{ack_req = ?ENABLED,
845         frame_type = ?FTYPE_DATA,
846         src_addr_mode = ?EXTENDED,
847         dest_addr_mode = ?EXTENDED},
848     MH1 = #mac_header{src_addr = ?node1_addr,
849         dest_addr = ?node2_addr},
850
851
852
853 ok = erpc:call(Node1, lowpan_api, tx, [<<BinMeshHeader1/binary, Frag1/
      bitstring>>, FC1, MH1]),
854
855 ct:pal("Incomplete payload sent from node1 to node2 to trigger a timeout"),
856 lowpan_node:stop_lowpan_node(Node1, Pid1).
857

```



```

858 %-----
859 % Receiver node 2 should experience a timeout
860 %-----
861 timeout_receiver(Config) ->
862     {Pid2, Node2} = ?config(node2, Config),
863     reassembly_timeout = erpc:call(Node2, lowpan_api, frameReception, []),
864     ct:pal("Timeout occurred~n"),
865     lowpan_node:stop_lowpan_node(Node2, Pid2).
866
867 %-----
868 % Send duplicate fragment to node 2
869 %-----
870 duplicate_sender(Config) ->
871     {Pid1, Node1} = ?config(node1, Config),
872
873     Data1 = <<"Hello ">>,
874     Data2 = <<"World!">>,
875     PayloadLen = byte_size(Data1) + byte_size(Data2),
876
877     FragHeader1 = #frag_header{
878         frag_type = ?FRAG1_DHTYPE,
879         datagram_size = PayloadLen,
880         datagram_tag = 25,
881         datagram_offset = 0
882     },
883     FragHeader2 = #frag_header{
884         frag_type = ?FRAGN_DHTYPE,
885         datagram_size = PayloadLen,
886         datagram_tag = 25,
887         datagram_offset = 1
888     },
889
890     Frag1 = lowpan_core:buildDatagramPckt(FragHeader1, Data1),
891     Frag2 = lowpan_core:buildDatagramPckt(FragHeader2, Data2),
892
893     MeshHeader =
894         #mesh_header{
895             v_bit = 0,
896             f_bit = 0,
897             hops_left = 14,
898             originator_address = ?node1_addr,
899             final_destination_address = ?node2_addr
900         },
901
902     BinMeshHeader = lowpan_core:buildMeshHeader(MeshHeader),
903
904     FC = #frame_control{ack_req = ?ENABLED,
905                       frame_type = ?FTYPE_DATA,
906                       src_addr_mode = ?EXTENDED,
907                       dest_addr_mode = ?EXTENDED},
908     MH = #mac_header{src_addr = ?node1_addr,
909                   dest_addr = ?node2_addr},
910
911     ok = erpc:call(Node1, lowpan_api, tx, [<<BinMeshHeader/binary, Frag1/bitstring
912     >>, FC, MH]),
913     ok = erpc:call(Node1, lowpan_api, tx, [<<BinMeshHeader/binary, Frag1/bitstring
914     >>, FC, MH]), % duplicated fragment
915     ok = erpc:call(Node1, lowpan_api, tx, [<<BinMeshHeader/binary, Frag2/bitstring
916     >>, FC, MH]),
917
918     ct:pal("Fragments sent from node1 and node2 to node3 with the same tag"),
919     lowpan_node:stop_lowpan_node(Node1, Pid1).

```

```

917
918 %-----
919 % Reception of payloads from node 1 and node 2 by node 3 with the same tag
920 %-----
921 duplicate_receiver(Config) ->
922     {Pid2, Node2} = ?config(node2, Config),
923
924     ReceivedData1 = erpc:call(Node2, lowpan_api, frameReception, []),
925
926     ExpectedData = <<"Hello World!">>,
927     %io:format("Expected: ~p~n~nReceived: ~p~n", [ExpectedData, ReceivedData1]),
928     ReceivedData1 = ExpectedData,
929     lowpan_node:stop_lowpan_node(Node2, Pid2).
930
931
932 %-----
933 % Send a packet that needs routing from node 1 to node 4
934 %-----
935 multiple_hop_sender(Config) ->
936     {Pid1, Node1} = ?config(node1, Config),
937     IPv6Pckt = ?config(ipv6_packet, Config),
938     ok = erpc:call(Node1, lowpan_api, sendPacket, [IPv6Pckt, false]),
939     ct:pal("multi hop packet sent successfully from node1 to node4"),
940     lowpan_node:stop_lowpan_node(Node1, Pid1).
941
942 %-----
943 % Reception of a routed packet
944 %-----
945 multiple_hop_receiver2(Config) ->
946     {Pid2, Node2} = ?config(node2, Config),
947     erpc:call(Node2, lowpan_api, frameReception, []),
948     lowpan_node:stop_lowpan_node(Node2, Pid2).
949
950
951 multiple_hop_receiver3(Config) ->
952     {Pid3, Node3} = ?config(node3, Config),
953     erpc:call(Node3, lowpan_api, frameReception, []),
954     lowpan_node:stop_lowpan_node(Node3, Pid3).
955
956 multiple_hop_receiver4(Config) ->
957     {Pid4, Node4} = ?config(node4, Config),
958     IPv6Pckt = ?config(ipv6_packet, Config),
959
960     {CompressedHeader, _} = lowpan_core:compressIpv6Header(IPv6Pckt, true),
961     PcktInfo = lowpan_core:getPcktInfo(IPv6Pckt),
962     Payload = PcktInfo#ipv6PckInfo.payload,
963     CompressedIpv6Packet = <<CompressedHeader/binary, Payload/bitstring>>,
964
965     ReceivedData = erpc:call(Node4, lowpan_api, frameReception, []),
966
967     io:format("Expected: ~p~n~nReceived: ~p~n", [CompressedIpv6Packet,
968         ReceivedData]),
969     ReceivedData = CompressedIpv6Packet,
970
971     ct:pal("Routed packet received successfully at node4"),
972     lowpan_node:stop_lowpan_node(Node4, Pid4).
973
974 %-----
975 % Send a none lowpan frame to node 2
976 %-----
977 nalp_sender(Config) ->

```

```

978     {Pid1, Node1} = ?config(node1, Config),
979     IPv6Pckt = ?config(ipv6_packet, Config),
980
981     Frame = <<?NALP_DHTYPE, IPv6Pckt/bitstring>>,
982
983     FC = #frame_control{ack_req = ?ENABLED,
984                       frame_type = ?FTYPE_DATA,
985                       src_addr_mode = ?EXTENDED,
986                       dest_addr_mode = ?EXTENDED},
987
988     MH = #mac_header{src_addr = ?node1_addr,
989                    dest_addr = ?node2_addr},
990
991     error_nalp = erpc:call(Node1, lowpan_api, tx, [Frame, FC, MH]),
992     ct:pal("NALP error correctly received"),
993     lowpan_node:stop_lowpan_node(Node1, Pid1).
994
995
996 %-----
997 % Send a broadcast packet
998 %-----
999 broadcast_sender(Config) ->
1000     {Pid1, Node1} = ?config(node1, Config),
1001     IPv6Pckt = ?config(ipv6_packet, Config),
1002     ok = erpc:call(Node1, lowpan_api, sendPacket, [IPv6Pckt, false]),
1003     ct:pal("Broadcast packet sent successfully"),
1004     lowpan_node:stop_lowpan_node(Node1, Pid1).
1005
1006 %-----
1007 % Reception of a broadcasted packet
1008 %-----
1009 broadcast_receiver(Config) ->
1010     {Pid2, Node2} = ?config(broadcast_node, Config),
1011     IPv6Pckt = ?config(ipv6_packet, Config),
1012
1013     {CompressedHeader, _} = lowpan_core:compressIpv6Header(IPv6Pckt, false),
1014     PcktInfo = lowpan_core:getPcktInfo(IPv6Pckt),
1015     Payload = PcktInfo#ipv6PckInfo.payload,
1016     CompressedIpv6Packet = <<CompressedHeader/binary, Payload/bitstring>>,
1017
1018     ReceivedData = erpc:call(Node2, lowpan_api, frameReception, []),
1019
1020     %io:format("Expected: ~p~n~nReceived: ~p~n", [CompressedIpv6Packet,
1021           ReceivedData]),
1022     ReceivedData = CompressedIpv6Packet,
1023
1024     ct:pal("Routed packet received successfully at node4"),
1025
1026     lowpan_node:stop_lowpan_node(Node2, Pid2).
1027
1028 %-----
1029 % Send a datagram with special hopsleft value 0xF
1030 %-----
1031 extended_hopsleft_sender(Config) ->
1032     {Pid1, Node1} = ?config(node1, Config),
1033     IPv6Pckt = ?config(ipv6_packet, Config),
1034     ok = erpc:call(Node1, lowpan_api, extendedHopsleftTx, [IPv6Pckt]),
1035     ct:pal("extended hop left packet sent successfully from node1 to node4"),
1036     lowpan_node:stop_lowpan_node(Node1, Pid1).
1037
1038 %-----

```

```

1039 % Discard datagram received from node 1
1040 %-----
1041 extended_hopsleft_receiver2(Config) ->
1042     {Pid2, Node2} = ?config(node2, Config),
1043     erpc:call(Node2, lowpan_api, frameReception, []),
1044     lowpan_node:stop_lowpan_node(Node2, Pid2).
1045
1046
1047 extended_hopsleft_receiver3(Config) ->
1048     {Pid3, Node3} = ?config(node3, Config),
1049     erpc:call(Node3, lowpan_api, frameReception, []),
1050     lowpan_node:stop_lowpan_node(Node3, Pid3).
1051
1052 extended_hopsleft_receiver4(Config) ->
1053     {Pid4, Node4} = ?config(node4, Config),
1054     IPv6Pckt = ?config(ipv6_packet, Config),
1055
1056     {CompressedHeader, _} = lowpan_core:compressIpv6Header(IPv6Pckt, true),
1057     PcktInfo = lowpan_core:getPcktInfo(IPv6Pckt),
1058     Payload = PcktInfo#ipv6PckInfo.payload,
1059     CompressedIpv6Packet = <<CompressedHeader/binary, Payload/bitstring>>,
1060
1061     ReceivedData = erpc:call(Node4, lowpan_api, frameReception, []),
1062
1063     %io:format("Expected: ~p~n~nReceived: ~p~n", [CompressedIpv6Packet,
1064         ReceivedData]),
1065     ReceivedData = CompressedIpv6Packet,
1066
1067     ct:pal("Routed packet received successfully at node4"),
1068     lowpan_node:stop_lowpan_node(Node4, Pid4).
1069 %-----
1070 % Send a packet in mesh level scope (mesh-local prefix used)
1071 %-----
1072 mesh_prefix_sender(Config) ->
1073     {Pid1, Node1} = ?config(node1, Config),
1074     IPv6Pckt = ?config(ipv6_packet, Config),
1075     ok = erpc:call(Node1, lowpan_api, sendPacket, [IPv6Pckt, false]),
1076     ct:pal("Broadcast packet sent successfully"),
1077     lowpan_node:stop_lowpan_node(Node1, Pid1).
1078
1079 %-----
1080 % Reception of a packet
1081 %-----
1082 mesh_prefix_receiver(Config) ->
1083     {Pid2, Node2} = ?config(node2, Config),
1084     IPv6Pckt = ?config(ipv6_packet, Config),
1085
1086     {CompressedHeader, _} = lowpan_core:compressIpv6Header(IPv6Pckt, false),
1087     PcktInfo = lowpan_core:getPcktInfo(IPv6Pckt),
1088     Payload = PcktInfo#ipv6PckInfo.payload,
1089     CompressedIpv6Packet = <<CompressedHeader/binary, Payload/bitstring>>,
1090
1091     ReceivedData = erpc:call(Node2, lowpan_api, frameReception, []),
1092
1093     %io:format("Expected: ~p~n~nReceived: ~p~n", [CompressedIpv6Packet,
1094         ReceivedData]),
1095     ReceivedData = CompressedIpv6Packet,
1096
1097     ct:pal("Routed packet received successfully at node4"),
1098     lowpan_node:stop_lowpan_node(Node2, Pid2).

```

A.11 Robot application code

```
1 -module(robot).
2
3 -behaviour(application).
4
5 -include("utils.hrl").
6
7 -export([
8     tx/0,
9     tx3/0,
10    tx4/0,
11    tx5/0,
12    tx_unc_ipv6/0,
13    tx_iphc_pkt/0,
14    tx_frag_iphc_pkt/0,
15    tx_big_payload/1,
16    tx_with_udp/0,
17    tx_msh_iphc_pkt/0,
18    tx_msh_frag_iphc_pkt/0,
19    msh_pkt_tx/0,
20    msh_big_pkt_tx/0,
21    rx/0,
22    tx_broadcast_pkt/0,
23    extendedHopsleftTx/0,
24    tx_unc_ipv6_udp/0,
25    tx_comp_ipv6_udp/0,
26    tx_mesh_prefix/0,
27    ieeeTx2/0,
28    ieeeTx3/0,
29    tx_big_payload3/1,
30    tx_big_payload4/1,
31    tx_big_payload5/1
32 ]).
33
34 -export([start/2]).
35 -export([stop/1]).
36
37
38 %--- Macros -----
39
40 -define(TX_ANTD, 16450).
41 -define(RX_ANTD, 16450).
42
43 %-----
44 % Sends uncompressed ipv6 packet format
45 %-----
46 tx_unc_ipv6() ->
47     Ipv6Pckt = ipv6:buildIpv6Packet(?IPv6Header, ?Payload),
48     io:format("Frame ~p~n", [Ipv6Pckt]),
49     io:format("Fragment size: ~p bytes~n", [byte_size(Ipv6Pckt)]),
50
51     lowpan_api:sendUncDatagram(Ipv6Pckt, ?FrameControl, ?MacHeader).
52
53
54 %-----
```

```

55 % Sends compressed header packet format
56 %-----
57 tx_iphc_pkt() ->
58     InlineData = <<12:8, ?Node1MacAddress/binary, ?Node2MacAddress/binary>>,
59     ExpectedHeader =
60         <<?IPHC_DHTYPE:3, 3:2, 12:1, 3:2, 0:1, 0:1, 1:2, 0:1, 0:1, 1:2, InlineData
61             /binary>>,
62
63     % Create the IPHC packet
64     IPHC = lowpan_core:createIphcPckt(ExpectedHeader, ?Payload),
65     io:format("IphcHeader ~p-n", [IPHC]),
66     io:format("Fragment size: ~p bytes-n", [byte_size(IPHC)]),
67
68     lowpan_api:tx(IPHC, ?FrameControl, ?MacHeader).
69 %-----
70 % Sends meshed and compressed header packet format
71 %-----
72 tx_msh_iphc_pkt() ->
73     Ipv6Pckt = ipv6:buildIpv6Packet(?IPv6Header, ?Payload),
74     {CompressedHeader, _} = lowpan_core:compressIpv6Header(Ipv6Pckt, true),
75
76     MeshHeader =
77         #mesh_header{
78             v_bit = 0,
79             f_bit = 0,
80             hops_left = 14,
81             originator_address = ?Node1MacAddress,
82             final_destination_address = ?Node2MacAddress
83         },
84
85     BinMeshHeader = lowpan_core:buildMeshHeader(MeshHeader),
86     Datagram = <<BinMeshHeader/binary, CompressedHeader/binary, ?Payload/bitstring
87         >>,
88     io:format("Datagram ~p-n", [Datagram]),
89
90     lowpan_api:tx(Datagram, ?FrameControl, ?MacHeader).
91 %-----
92 % Sends fragmented and compressed packet format
93 %-----
94 tx_frag_iphc_pkt() ->
95     Ipv6Pckt = ipv6:buildIpv6Packet(?IPv6Header, ?Payload),
96     {CompressedHeader, _} = lowpan_core:compressIpv6Header(Ipv6Pckt, false),
97     PcktLen = byte_size(Ipv6Pckt),
98
99     FragHeader =
100         #frag_header{
101             frag_type = ?FRAG1_DHTYPE,
102             datagram_size = PcktLen,
103             datagram_tag = 124
104         },
105
106     FragHeaderBin = lowpan_core:buildFirstFragHeader(FragHeader),
107
108     Datagram = <<FragHeaderBin/binary, CompressedHeader/binary, ?Payload/bitstring
109         >>,
110     io:format("Frame ~p-n", [Datagram]),
111     io:format("Fragment size: ~p bytes-n", [byte_size(Datagram)]),
112
113     lowpan_api:tx(Datagram, ?FrameControl, ?MacHeader).

```

```

114 %-----
115 % Sends meshed, fragmented and compressed packet format
116 %-----
117 tx_msh_frag_iphc_pckt() ->
118     Ipv6Pckt = ipv6:buildIpv6Packet(?IPv6Header, ?Payload),
119     {CompressedHeader, _} = lowpan_core:compressIpv6Header(Ipv6Pckt, true),
120     PcktLen = byte_size(Ipv6Pckt),
121
122     FragHeader =
123         #frag_header{
124             frag_type = ?FRAG1_DHTYPE,
125             datagram_size = PcktLen,
126             datagram_tag = 124
127         },
128
129     FragHeaderBin = lowpan_core:buildFirstFragHeader(FragHeader),
130
131     MeshHeader =
132         #mesh_header{
133             v_bit = 0,
134             f_bit = 0,
135             hops_left = 14,
136             originator_address = ?Node1MacAddress,
137             final_destination_address = ?Node2MacAddress
138         },
139
140     BinMeshHeader = lowpan_core:buildMeshHeader(MeshHeader),
141     Datagram =
142         <<BinMeshHeader/binary, FragHeaderBin/binary, CompressedHeader/binary, ?
143             Payload/bitstring>>,
144     io:format("Datagram ~p~n", [Datagram]),
145
146     lowpan_api:tx(Datagram, ?FrameControl, ?MacHeader).
147 %-----
148 % Sends broadcast packet format
149 %-----
150 tx_broadcast_pckt() ->
151     Ipv6Pckt = ipv6:buildIpv6Packet(?IPv6Header, ?Payload),
152     {CompressedHeader, _} = lowpan_core:compressIpv6Header(Ipv6Pckt, false),
153     PcktLen = byte_size(Ipv6Pckt),
154
155     FragHeader =
156         #frag_header{
157             frag_type = ?FRAG1_DHTYPE,
158             datagram_size = PcktLen,
159             datagram_tag = 124
160         },
161
162     FragHeaderBin = lowpan_core:buildFirstFragHeader(FragHeader),
163
164     DestMacAddr = lowpan_core:generateEUI64MacAddr(<<16#1234:16>>),
165
166     DestAddr = <<16#FF02:16, 0:64, 1:16, 16#FF00:16, 16#1234:16>>,
167     DestAddress = binary:decode_unsigned(DestAddr),
168     {_, BroadcastHeader, _} = lowpan_core:getNextHop(?Node1MacAddress, ?
169         Node1MacAddress, DestMacAddr, DestAddress, 3, false),
170
171     Datagram =
172         <<BroadcastHeader/binary, FragHeaderBin/binary, CompressedHeader/binary, ?
173             Payload/bitstring>>,
174     io:format("Datagram ~p~n", [Datagram]),

```

```

173
174     MacHeader = #mac_header{src_addr = ?Node1MacAddress, dest_addr = DestMacAddr},
175     lowpan_api:tx(Datagram, ?FrameControl, MacHeader).
176
177
178 %-----
179 % Simple transmission to node 2
180 %-----
181 tx() ->
182     Ipv6Pckt = ipv6:buildIpv6Packet(?IPv6Header, ?Payload),
183     lowpan_api:sendPacket(Ipv6Pckt, true).
184
185 %-----
186 % Simple transmission to node 3
187 %-----
188 tx3() ->
189     Ipv6Pckt = ipv6:buildIpv6Packet(?IPv6Header3, ?Payload),
190     lowpan_api:sendPacket(Ipv6Pckt, true).
191
192 %-----
193 % Simple transmission to node 4
194 %-----
195 tx4() ->
196     Ipv6Pckt = ipv6:buildIpv6Packet(?IPv6Header4, ?Payload),
197     lowpan_api:sendPacket(Ipv6Pckt, true).
198
199 %-----
200 % Simple transmission to node 5
201 %-----
202 tx5() ->
203     Ipv6Pckt = ipv6:buildIpv6Packet(?IPv6Header5, ?Payload),
204     lowpan_api:sendPacket(Ipv6Pckt, true).
205
206
207
208 %-----
209 % Big payload transmission, N = nbr of chunk in payload
210 %-----
211 tx_big_payload(N) ->
212     Payload = lowpan_core:generateChunks(N),
213
214     Node1Address = lowpan_core:generateLLAddr(?Node1MacAddress),
215     Node2Address = lowpan_core:generateLLAddr(?Node2MacAddress),
216     PayloadLength = byte_size(Payload),
217
218     IPv6Header =
219         #ipv6_header{
220             version = 6,
221             traffic_class = 0,
222             flow_label = 0,
223             payload_length = PayloadLength,
224             next_header = 12,
225             hop_limit = 64,
226             source_address = Node1Address,
227             destination_address = Node2Address
228         },
229     Ipv6Pckt = ipv6:buildIpv6Packet(IPv6Header, Payload),
230     lowpan_api:sendPacket(Ipv6Pckt, true).
231
232 %% Tx to node 3
233 tx_big_payload3(N) ->
234     Payload = lowpan_core:generateChunks(N),

```



```

235
236 Node1Address = lowpan_core:generateLLAddr(?Node1MacAddress),
237 Node2Address = lowpan_core:generateLLAddr(?Node3MacAddress),
238 PayloadLength = byte_size(Payload),
239
240 IPv6Header =
241     #ipv6_header{
242         version = 6,
243         traffic_class = 0,
244         flow_label = 0,
245         payload_length = PayloadLength,
246         next_header = 12,
247         hop_limit = 64,
248         source_address = Node1Address,
249         destination_address = Node2Address
250     },
251 Ipv6Pckt = ipv6:buildIpv6Packet(IPv6Header, Payload),
252 lowpan_api:sendPacket(Ipv6Pckt, true).
253
254 %% Tx to node 4
255 tx_big_payload4(N) ->
256     Payload = lowpan_core:generateChunks(N),
257
258     Node1Address = lowpan_core:generateLLAddr(?Node1MacAddress),
259     Node2Address = lowpan_core:generateLLAddr(?Node4MacAddress),
260     PayloadLength = byte_size(Payload),
261
262     IPv6Header =
263         #ipv6_header{
264             version = 6,
265             traffic_class = 0,
266             flow_label = 0,
267             payload_length = PayloadLength,
268             next_header = 12,
269             hop_limit = 64,
270             source_address = Node1Address,
271             destination_address = Node2Address
272         },
273     Ipv6Pckt = ipv6:buildIpv6Packet(IPv6Header, Payload),
274     lowpan_api:sendPacket(Ipv6Pckt, true).
275
276 %% Tx to node 5
277 tx_big_payload5(N) ->
278     Payload = lowpan_core:generateChunks(N),
279
280     Node1Address = lowpan_core:generateLLAddr(?Node1MacAddress),
281     Node2Address = lowpan_core:generateLLAddr(?Node5MacAddress),
282     PayloadLength = byte_size(Payload),
283
284     IPv6Header =
285         #ipv6_header{
286             version = 6,
287             traffic_class = 0,
288             flow_label = 0,
289             payload_length = PayloadLength,
290             next_header = 12,
291             hop_limit = 64,
292             source_address = Node1Address,
293             destination_address = Node2Address
294         },
295     Ipv6Pckt = ipv6:buildIpv6Packet(IPv6Header, Payload),
296     lowpan_api:sendPacket(Ipv6Pckt, true).

```

```

297
298
299 %-----
300 % Transmission of uncompressed ipv6 packet with udp next header
301 %-----
302 tx_unc_ipv6_udp() ->
303     Payload = <<"Hello world">>,
304     PayloadLength = byte_size(Payload),
305     IPv6Header =
306         #ipv6_header{
307             version = 6,
308             traffic_class = 0,
309             flow_label = 0,
310             % 4 bytes for the UDP header
311             payload_length = PayloadLength,
312             next_header = 17,
313             hop_limit = 255,
314             source_address = ?Node1Address,
315             destination_address = ?Node2Address
316         },
317     UdpHeader =
318         #udp_header{
319             source_port = 1025,
320             destination_port = 61617,
321             length = PayloadLength,
322             checksum = 16#f88c
323         },
324
325     Ipv6Pckt = ipv6:buildIpv6UdpPacket(IPv6Header, UdpHeader, Payload),
326     io:format("Frame ~p~n", [Ipv6Pckt]),
327     io:format("Fragment size: ~p bytes~n", [byte_size(Ipv6Pckt)]),
328
329     lowpan_api:sendUncDatagram(Ipv6Pckt, ?FrameControl, ?MacHeader).
330
331
332 %-----
333 % Transmission of compressed ipv6 packet with udp next header
334 %-----
335 tx_comp_ipv6_udp() ->
336     Payload = <<"Hello world">>,
337     PayloadLength = byte_size(Payload),
338     IPv6Header =
339         #ipv6_header{
340             version = 6,
341             traffic_class = 0,
342             flow_label = 0,
343             % 4 bytes for the UDP header
344             payload_length = PayloadLength,
345             next_header = 17,
346             hop_limit = 64,
347             source_address = ?Node1Address,
348             destination_address = ?Node2Address
349         },
350     UdpHeader =
351         #udp_header{
352             source_port = 1025,
353             destination_port = 61617,
354             length = PayloadLength,
355             checksum = 16#f88c
356         },
357
358     Ipv6Pckt = ipv6:buildIpv6UdpPacket(IPv6Header, UdpHeader, Payload),

```

```

359     lowpan_api:sendPacket(Ipv6Pckt).
360
361 %-----
362 % Ipv6 with nextHeader packet format verification
363 %-----
364 tx_with_udp() ->
365     IPv6Header =
366         #ipv6_header{
367             version = 6,
368             traffic_class = 0,
369             flow_label = 0,
370             % 4 bytes for the UDP header
371             payload_length = ?PayloadLength,
372             next_header = 17,
373             hop_limit = 64,
374             source_address = ?Node1Address,
375             destination_address = ?Node2Address
376         },
377     UdpHeader =
378         #udp_header{
379             source_port = 1025,
380             destination_port = 61617,
381             length = ?PayloadLength,
382             checksum = 16#f88c
383         },
384
385     Ipv6Pckt = ipv6:buildIpv6UdpPacket(IPv6Header, UdpHeader, ?Payload),
386     lowpan_api:sendPacket(Ipv6Pckt).
387
388 %-----
389 % Transmission of packet that needs routing
390 %-----
391 msh_pckt_tx() ->
392     IPv6Header =
393         #ipv6_header{
394             version = 6,
395             traffic_class = 0,
396             flow_label = 0,
397             payload_length = ?PayloadLength,
398             next_header = 10,
399             hop_limit = 64,
400             source_address = ?Node1Address,
401             destination_address = ?Node3Address
402         },
403
404     Ipv6Pckt = ipv6:buildIpv6Packet(IPv6Header, ?Payload),
405     lowpan_api:sendPacket(Ipv6Pckt).
406
407 %-----
408 % Transmission of big packet that needs routing
409 %-----
410 msh_big_pckt_tx() ->
411     IPv6Header =
412         #ipv6_header{
413             version = 6,
414             traffic_class = 0,
415             flow_label = 0,
416             payload_length = ?PayloadLength,
417             next_header = 10,
418             hop_limit = 64,
419             source_address = ?Node1Address,
420             destination_address = ?Node3Address

```

```

421     },
422
423     Ipv6Pckt = ipv6:buildIpv6Packet(IPv6Header, ?BigPayload),
424     lowpan_api:sendPacket(Ipv6Pckt).
425
426 %-----
427 % Extended hopsLeft packet transmission
428 %-----
429 extendedHopsleftTx() ->
430     Ipv6Pckt = ipv6:buildIpv6Packet(?IPv6Header, ?Payload),
431     lowpan_api:extendedHopsleftTx(Ipv6Pckt).
432
433 %-----
434 % Transmission of mesh level packet (mesh-local prefix used)
435 %-----
436 tx_mesh_prefix() ->
437     MacAddress = lowpan_core:generateEUI64MacAddr(?Node2MacAddress),
438     IPv6Header = #ipv6_header{
439         version = 6,
440         traffic_class = 0,
441         flow_label = 0,
442         payload_length = byte_size(?Payload),
443         next_header = 12,
444         hop_limit = 64,
445         source_address = ?Node1Address,
446         destination_address = <<?MESH_LOCAL_PREFIX:16, 16#0DB8:16, 0:32,
447             2:8,0:48, MacAddress/binary>>
448     },
449     Packet = ipv6:buildIpv6Packet(IPv6Header, ?Payload),
450     lowpan_api:sendPacket(Packet).
451
452 %-----
453 % Data reception
454 %-----
455 rx() ->
456     grisp_led:color(2, red),
457     lowpan_api:frameReception(),
458     grisp_led:color(2, green),
459     rx().
460
461 %-----
462 % Direct ieee data transmission to node 2
463 %-----
464 ieeeTx2()->
465     FrameControl = #frame_control{
466         frame_type = ?FTYPE_DATA,
467         src_addr_mode = ?EXTENDED,
468         dest_addr_mode = ?EXTENDED},
469
470     MacHeader = #mac_header{src_addr = ?Node1MacAddress,
471         dest_addr = ?Node2MacAddress},
472
473     lowpan_api:tx(<<"Hello">>, FrameControl, MacHeader).
474
475 %-----
476 % Direct ieee data transmission to node 3
477 %-----
478 ieeeTx3()->
479     FrameControl = #frame_control{
480         frame_type = ?FTYPE_DATA,
481         src_addr_mode = ?EXTENDED,
482         dest_addr_mode = ?EXTENDED},

```

```

482
483     MacHeader = #mac_header{src_addr = ?Node1MacAddress,
484                   dest_addr = ?Node3MacAddress},
485
486     lowpan_api:tx(<<"Hello">>, FrameControl, MacHeader).
487
488
489 %-----
490 % IEEE 802.15.4 setup only for manual configuration
491 %-----
492 % ieee802154_setup(MacAddr)->
493 %     ieee802154:start(#ieee_parameters{
494 %         duty_cycle = duty_cycle_non_beacon,
495 %         input_callback = fun lowpan_api:input_callback/4
496 %     }),
497
498 %     case application:get_env(robot, pan_id) of
499 %     {ok, PanId} ->
500 %         ieee802154:set_pib_attribute(mac_pan_id, PanId);
501 %     _ ->
502 %         ok
503 %     end,
504
505 %     case byte_size(MacAddr) of
506 %     ?EXTENDED_ADDR_LEN -> ieee802154:set_pib_attribute(mac_extended_address,
507 %         MacAddr);
508 %     ?SHORT_ADDR_LEN -> ieee802154:set_pib_attribute(mac_short_address,
509 %         MacAddr)
510 %     end,
511
512 %     ieee802154:rx_on().
513
514 start(_Type, _Args) ->
515 {ok, Supervisor} = robot_sup:start_link(),
516 grisp:add_device(spi2, pmod_uwb),
517 pmod_uwb:write(tx_antd, #{tx_antd => ?TX_ANTD}),
518 pmod_uwb:write(lde_if, #{lde_rxantd => ?RX_ANTD}),
519
520 NodeMacAddr = case application:get_env(robot, mac_addr) of
521 {ok, MacAddr} ->
522     MacAddr;
523 _ ->
524     ?Node1MacAddress
525 end,
526
527 %ieee802154_setup(NodeMacAddr),
528
529 lowpan_api:start(#{node_mac_addr => NodeMacAddr, routing_table => ?
530     Default_routing_table}),
531
532 %rx(),
533 {ok, Supervisor}.
534
535 % @private
536 stop(_State) ->
537 ok.

```

A.12 Mockups files code

```
1 -module(mock_mac).
2
3 -include("../src/mac_frame.hrl").
4
5 -include_lib("eunit/include/eunit.hrl").
6
7 -behaviour(gen_mac_layer).
8
9 -export([send_data/3]).
10 % -export([send_data/4]).
11
12 % -export([reception/1]).
13 -export([reception/0]).
14 %% gen_server callbacks
15 -export([init/1]).
16 -export([tx/4]).
17 -export([rx/1]).
18 -export([rx_on/2]).
19 -export([rx_off/1]).
20 -export([set/3]).
21 -export([get/2]).
22 -export([terminate/2]).
23
24 % --- API -----
25 reception() ->
26     mac_layer_behaviour:rx().
27
28 send_data(FrameControl, MacHeader, Payload) ->
29     mac_layer_behaviour:tx(FrameControl, MacHeader, Payload).
30
31 % --- Callbacks -----
32 init(Params) ->
33     PhyModule =
34         case Params of
35             #{phy_layer := PHY} ->
36                 PHY;
37             _ ->
38                 pmod_uwb
39         end,
40     #{
41         phy_layer => PhyModule,
42         rx => off,
43         pib =>
44             #{
45                 mac_pan_id => <<16#FFFF:16>>,
46                 mac_extended_address => <<16#FFFFFFFF:64>>,
47                 mac_short_address => <<16#FFFF:16>>
48             }
49     }.
50
51 tx(
52     State,
53     #frame_control{ack_req = ?ENABLED} = FrameControl,
54     #mac_header{seqnum = Seqnum} = MacHeader,
55     Payload
56 ) ->
57     Frame = mac_frame:encode(FrameControl, MacHeader, Payload),
58     transmission(Frame),
```

```

59     _RxFrame = receive_ack(Seqnum),
60     {ok, State};
61 tx(State, FrameControl, MacHeader, Payload) ->
62     Frame = mac_frame:encode(FrameControl, MacHeader, Payload),
63     {transmission(Frame), State}.
64
65 rx(State) ->
66     {ok, State, receive_()}.
67
68 rx_on(State, _) ->
69     {ok, State#{rx => on}}.
70
71 rx_off(State) ->
72     {ok, State#{rx => off}}.
73
74 get("#{pib := PIB} = State, Attribute) ->
75     case maps:get(Attribute, PIB) of
76     {badkey, _} ->
77         {error, State, unsupported_attribute};
78     Val ->
79         {ok, State, Val}
80     end.
81
82 set("#{pib := PIB} = State, Attribute, Value) ->
83     case maps:update(Attribute, Value, PIB) of
84     {badkey, _} ->
85         {error, State, unsupported_attribute};
86     NewPIB ->
87         {ok, State#{pib => NewPIB}}
88     end.
89
90 terminate(_State, _Reason) ->
91     ok.
92
93 % --- Internals -----
94
95 receive_() ->
96     FrameControl = #frame_control{pan_id_compr = ?ENABLED, frame_version = 2#00},
97     MacHeader =
98         #mac_header{
99             seqnum = 0,
100             dest_pan = <<16#DECA:16>>,
101             dest_addr = <<"RX">>,
102             src_pan = <<16#DECA:16>>,
103             src_addr = <<"TX">>
104         },
105     {FrameControl, MacHeader, <<"Hello">>}.
106
107 % Received MAC frame for an ACK is only composed of the Frame control, the seqnum
108 % and the FCS
109 receive_ack(Seqnum) ->
110     FrameControl = #frame_control{frame_type = ?FTYPE_ACK},
111     MacHeader = #mac_header{seqnum = Seqnum},
112     {FrameControl, MacHeader, <<>>}.
113
114 transmission(Frame) when byte_size(Frame) < 125 ->
115     io:format("~w~n", [Frame]),
116     ok.

```

```

1 -module(mock_phy_network).
2 -behaviour(gen_statem).

```

```

3
4 -include("../src/mac_frame.hrl").
5
6
7 -export([start_link/2]).
8 -export([start/2]).
9 -export([stop_link/0]).
10
11 -export([transmit/2]).
12 -export([reception/0]).
13 -export([reception_async/0]).
14 -export([reception/1]).
15 -export([disable_rx/0]).
16
17 -export([set_frame_timeout/1]).
18 -export([set_preamble_timeout/1]).
19 -export([disable_preamble_timeout/0]).
20
21 -export([suspend_frame_filtering/0]).
22 -export([resume_frame_filtering/0]).
23
24 -export([read/1]).
25 -export([write/2]).
26
27 -export([rx_ranging_info/0]).
28 -export([signal_power/0]).
29 -export([rx_preamble_repetition/0]).
30 -export([rx_data_rate/0]).
31 -export([prf_value/0]).
32 -export([get_conf/0]).
33 -export([get_rx_metadata/0]).
34
35 %% gen_statem callbacks
36 -export([init/1]).
37 -export([callback_mode/0]).
38 -export([idle/3, rx_on/3, idle_rx/3, idle_to/3]).
39 -export([terminate/2]).
40
41 %--- Records -----
42
43
44
45 %--- API -----
46
47 start_link(_Connector, Params) ->
48     gen_statem:start_link({local, ?MODULE}, ?MODULE, Params, []).
49
50 start(_Connector, Params) ->
51     gen_statem:start({local, ?MODULE}, ?MODULE, Params, []).
52
53 stop_link() ->
54     gen_statem:stop(?MODULE).
55
56 transmit(Frame, Options) ->
57     gen_statem:call(?MODULE, {transmit, Frame, Options}).
58
59 reception() ->
60     case {read(drx_conf), read(rx_fwto)} of
61     #{drx_pretoc := 0}, #{rxfwto := RXFWTO}} ->
62         rx_(round(RXFWTO/1000), rxrfto);
63     #{drx_pretoc := PRETOC}, _ ->
64         rx_(round(PRETOC/1000), rxpto)

```



```

65     end.
66
67 reception_async() ->
68     case reception() of
69         {error, _} = Err ->
70             %ct:log("Error? : ~p", [Err]),
71             Err;
72         Frame ->
73             %ct:log("Frame: ~p", [Frame]),
74             Metadata = get_rx_metadata(),
75             ieee802154_events:rx_event(Frame, Metadata)
76     end.
77
78 rx_(Timeout, TimeoutError) ->
79     case gen_statem:call(?MODULE, {enable_rx, Timeout}, infinity) of
80         timeout ->
81             {error, TimeoutError};
82         affrej ->
83             {error, affrej};
84         Ret ->
85             %ct:log("ret: ~p", [Ret]),
86             Ret
87     end.
88
89 reception(_RxOpts) ->
90     reception().
91
92 disable_rx() ->
93     gen_statem:call(?MODULE, {disable_rx}).
94
95 set_frame_timeout(Timeout) when is_float(Timeout) ->
96     set_frame_timeout(trunc(Timeout));
97 set_frame_timeout(Timeout) when is_integer(Timeout) ->
98     write(rx_fwto, #{rxfwto => Timeout}),
99     write(sys_cfg, #{rxwtoe => 2#1}). % enable receive wait timeout
100
101 set_preamble_timeout(Timeout) ->
102     write(drx_conf, #{drx_pretoc => Timeout}).
103
104 disable_preamble_timeout() ->
105     write(drx_conf, #{drx_pretoc => 0}).
106
107 suspend_frame_filtering() ->
108     write(sys_cfg, #{fffen => 0}).
109
110 resume_frame_filtering() ->
111     write(sys_cfg, #{fffen => 1}).
112
113 read(Reg) ->
114     gen_statem:call(?MODULE, {read, Reg}).
115
116 write(Reg, Value) ->
117     gen_statem:call(?MODULE, {write, Reg, Value}).
118
119 %--- API: Getters -----
120 rx_ranging_info() ->
121     #{rng := RNG} = read(rx_finfo),
122     RNG.
123
124 %% @doc Returns the estimated value of the signal power in dBm
125 %% cf. user manual section 4.7.2
126 signal_power() ->

```

```

127 C = channel_impulse_resp_pow() , % Channel impulse response power value (
    CIR_PWR)
128 A = case prf_value() of
129     16 -> 113.77;
130     64 -> 121.74
131 end, % Constant. For PRF of 16 MHz = 113.77, for PRF of 64MHz = 121.74
132 N = preamble_acc(), % Preamble accumulation count value (RXPACC but might be
    adjusted)
133 Num = C* math:pow(2, 17),
134 Dem = math:pow(N, 2),
135 Log = math:log10(Num / Dem),
136 10 * Log - A.
137
138 preamble_acc() ->
139     #{rxpacc := RXPACC} = read(rx_finfo),
140     #{rxpacc_nosat := RXPACC_NOSAT} = read(drx_conf),
141     if
142         RXPACC == RXPACC_NOSAT -> RXPACC;
143         true -> RXPACC - 5
144     end.
145
146 channel_impulse_resp_pow() ->
147     #{cir_pwr := CIR_POW} = read(rx_fqual),
148     CIR_POW.
149
150 %% @doc Gives the value of the PRF in MHz
151 -spec prf_value() -> 16 | 64.
152 prf_value() ->
153     #{agc_tune1 := AGC_TUNE1} = read(agc_ctrl),
154     case AGC_TUNE1 of
155         16#8870 -> 16;
156         16#889B -> 64
157     end.
158
159 %% @doc returns the preamble symbols repetition
160 rx_preamble_repetition() ->
161     #{rxpsr := RXPSR} = read(rx_finfo),
162     case RXPSR of
163         0 -> 16;
164         1 -> 64;
165         2 -> 1024;
166         3 -> 4096
167     end.
168
169 %% @doc returns the data rate of the received frame in kbps
170 rx_data_rate() ->
171     #{rxbr := RXBR} = read(rx_finfo),
172     case RXBR of
173         0 -> 110;
174         1 -> 850;
175         3 -> 6800
176     end.
177
178 get_conf() ->
179     gen_server:call(?MODULE, {get_conf}).
180
181 get_rx_metadata() ->
182     #{rng := Rng} = read(rx_finfo),
183     #{rx_stamp := RxStamp} = read(rx_time),
184     #{tx_stamp := TxStamp} = read(tx_time),
185     #{rxtofs := Rxtofs} = read(rx_ttcko),
186     #{rxttcki := Rxttcki} = read(rx_tttcki),

```

```

187     #{snr => snr(),
188       prf => prf_value(),
189       pre => rx_preamble_repetition(),
190       data_rate => rx_data_rate(),
191       rng => Rng,
192       rx_stamp => RxStamp,
193       tx_stamp => TxStamp,
194       rxtofs => Rxtofs,
195       rxttcki => Rxttcki}.
196
197 % Source: https://forum.qorvo.com/t/how-to-calculate-the-signal-to-noise-ratio-snr-of-dw1000/5585/3
198 snr() ->
199     Delta = 87-7.5,
200     RSL = signal_power(),
201     RSL + Delta.
202
203 %--- Internal: gen server callbacks -----
204
205 init("#{network := NetworkNode}) ->
206     {network_loop, NetworkNode} ! {register, node()},
207     ets:new(callback_table, [public, named_table]),
208     {ok, idle, #{regs => pmod_uwb_registers:default(),
209                network => NetworkNode,
210                conf => #phy_cfg{}}}.
211
212 callback_mode() ->
213     [state_functions, state_enter].
214
215 idle(enter, _OldState, Data) ->
216     {keep_state, Data};
217 idle({call, From}, {transmit, Frame, Options}, Data) ->
218     #{network := NetworkNode, regs := Regs} = Data,
219     NewRegs = tx(Frame, Options, NetworkNode, Regs),
220     case Options#tx_opts.wait4resp of
221     ?ENABLED ->
222         {next_state, rx_on, Data#{regs := NewRegs}, {reply, From, ok}};
223     ?DISABLED ->
224         {keep_state, Data#{regs := NewRegs}, {reply, From, ok}}
225     end;
226 idle({call, From}, {enable_rx, Timeout}, Data) ->
227     {next_state, rx_on, Data#{timeout => Timeout, waiting => From}};
228 idle({call, From}, {disable_rx}, Data) ->
229     {keep_state, Data, {reply, From, ok}};
230 idle(EventType, EventContent, Data) ->
231     handle_event(EventType, EventContent, Data).
232
233 rx_on(enter, _OldState, #{regs := Regs, timeout := Timeout} = Data) ->
234     NewRegs = enable_rx(Regs),
235     TimerRef = erlang:start_timer(Timeout, ?MODULE, rx_timeout),
236     {keep_state, Data#{regs => NewRegs, timer => TimerRef}};
237 rx_on({call, From}, {enable_rx, _Timeout}, Data) ->
238     {keep_state, Data#{waiting => From}}; % Happens when W4R is enabled
239 rx_on({call, From}, {disable_rx}, #{regs := Regs, timer := TimerRef} = Data) ->
240     erlang:cancel_timer(TimerRef),
241     NewRegs = pmod_uwb_registers:update_reg(Regs, sys_ctrl, #{rxenab => ?DISABLED}
242     ),
243     {next_state, idle, Data#{regs := NewRegs}, {reply, From, ok}};
244 rx_on(info, {frame, Frame}, #{timer := TimerRef, regs := Regs, network :=
245     NetworkNode} = Data) ->
246     erlang:cancel_timer(TimerRef),
247     %ct:log("Received frame: ~p", [Frame]),

```

```

246     NewRegs = handle_rx(Frame, NetworkNode, Regs),
247     case Data#{regs := NewRegs} of
248     #{waiting := From, regs:= #{sys_status := #{affrej := 0}}} ->
249         #{rx_buffer := RawFrame} = pmod_uwb_registers:get_value(NewRegs,
250             rx_buffer),
251         Reply = {byte_size(RawFrame), RawFrame},
252         NewData = maps:remove(waiting, Data),
253         {next_state, idle, NewData#{regs => NewRegs}, {reply, From, Reply}};
254     #{waiting := From, regs:= #{sys_status := #{affrej := 1}}} ->
255         NewData = maps:remove(waiting, Data),
256         {next_state, idle, NewData#{regs => NewRegs}, {reply, From, affrej}};
257     _ ->
258         {next_state, idle_rx, Data#{regs => NewRegs}}
259     end;
260 rx_on(info, {timeout, _, rx_timeout}, #{regs := Regs} = Data) ->
261     NewRegs = handle_timeout(Regs),
262     case Data of
263     #{waiting := From} ->
264         NewData = maps:remove(waiting, Data),
265         {next_state, idle, NewData#{regs => NewRegs}, {reply, From, timeout}};
266     _ ->
267         {next_state, idle_to, Data#{regs => NewRegs}}
268     end;
269 rx_on(EventType, EventContent, Data) ->
270     handle_event(EventType, EventContent, Data).
271 idle_rx(enter, _OldState, Data) ->
272     {keep_state, Data};
273 idle_rx({call, From}, {enable_rx, _Timeout}, #{regs := Regs} = Data) ->
274     case pmod_uwb_registers:get_value(Regs, sys_status) of
275     #{affrej := 1} ->
276         {next_state, idle, Data, {reply, From, affrej}};
277     _ ->
278         #{rx_buffer := RawFrame} = pmod_uwb_registers:get_value(Regs,
279             rx_buffer),
280         Reply = {byte_size(RawFrame), RawFrame},
281         NewData = maps:remove(waiting, Data),
282         {next_state, idle, NewData#{regs => Regs}, {reply, From, Reply}}
283     end;
284 idle_rx(EventType, EventContent, Data) ->
285     handle_event(EventType, EventContent, Data).
286 idle_to(enter, _OldState, Data) ->
287     {keep_state, Data};
288 idle_to({call, From}, {enable_rx, _Timeout}, Data) ->
289     {next_state, idle, Data, {reply, From, timeout}}.
290
291 handle_event({call, From}, {read, Reg}, #{regs := Regs} = Data) ->
292     Val = pmod_uwb_registers:get_value(Regs, Reg),
293     {keep_state, Data, {reply, From, Val}};
294 handle_event({call, From}, {write, Reg, Value}, #{regs := Regs} = Data) ->
295     NewRegs = pmod_uwb_registers:update_reg(Regs, Reg, Value),
296     {keep_state, Data#{regs => NewRegs}, {reply, From, ok}};
297 handle_event({call, From}, {get_conf}, #{conf := Conf} = Data) ->
298     {keep_state, Data, {reply, From, Conf}};
299 handle_event(info, Event, Data) ->
300     %ct:log("Event skipped: ~p", [Event]),
301     {keep_state, Data};
302 handle_event(EventType, EventContent, _Data) ->
303     error({unknown_event, EventType, EventContent}).
304
305 terminate(Reason, _) ->

```

```

306     io:format("Terminate: ~w", [Reason]).
307
308 %--- Internal -----
309
310 tx(Frame, Options, NetworkNode, Regs) ->
311     Rng = Options#tx_opts.ranging,
312     PhyFrame = {Rng, Frame},
313     %ct:log("Tx frame ~p", [Frame]),
314     {network_loop, NetworkNode} ! {tx, node(), PhyFrame},
315     pmod_uwb_registers:update_reg(Regs, tx_fctrl, #{tr => Rng}).
316
317 enable_rx(Regs) ->
318     NewRegs = pmod_uwb_registers:update_reg(Regs, sys_ctrl, #{rxenab => 1}),
319     pmod_uwb_registers:update_reg(NewRegs, sys_status, #{rxfcg => 0, affrej => 0})
320
321 handle_rx(⟦_, <<_:5/bitstring, ?FTYPE_ACK:3, _/bitstring>>=RawFrame], _, Regs) ->
322     %ct:log("Received Ack"),
323     NewRegs1 = pmod_uwb_registers:update_reg(Regs, sys_cfg, #{rxenab => ?DISABLED}
324     ),
325     pmod_uwb_registers:update_reg(NewRegs1, rx_buffer, #{rx_buffer => RawFrame});
326 handle_rx(Frame, NetworkNode, #{sys_cfg := #{iffen := ?ENABLED}} = Regs) ->
327     {Rng, RawFrame} = Frame,
328     #{short_addr := ShortAddress} = pmod_uwb_registers:get_value(Regs, panadr),
329     #{eui := ExtAddress} = pmod_uwb_registers:get_value(Regs, eui),
330     case check_address(RawFrame, ShortAddress, ExtAddress) of
331     ok ->
332         AckRegs = ack_reply(RawFrame, NetworkNode, Regs),
333         NewRegs = pmod_uwb_registers:update_reg(AckRegs, rx_finfo, #{rng =>
334         Rng}),
335         NewRegs1 = pmod_uwb_registers:update_reg(NewRegs, sys_cfg, #{rxenab =>
336         ?DISABLED}),
337         pmod_uwb_registers:update_reg(NewRegs1, rx_buffer, #{rx_buffer =>
338         RawFrame});
339     _ ->
340         NewRegs = pmod_uwb_registers:update_reg(Regs, sys_cfg, #{rxenab => ?
341         DISABLED}),
342         pmod_uwb_registers:update_reg(NewRegs, sys_status, #{affrej => 1})
343     end;
344 handle_rx(Frame, _, Regs) ->
345     {Rng, RawFrame} = Frame,
346     NewRegs = pmod_uwb_registers:update_reg(Regs, rx_finfo, #{rng => Rng}),
347     NewRegs1 = pmod_uwb_registers:update_reg(NewRegs, sys_cfg, #{rxenab => ?
348     DISABLED}),
349     pmod_uwb_registers:update_reg(NewRegs1, rx_buffer, #{rx_buffer => RawFrame}).
350
351 handle_timeout(Regs) ->
352     pmod_uwb_registers:update_reg(Regs, sys_status, #{rxenab => 0}).
353
354 check_address(Frame, ShortAddress, ExtAddress) -> % This will need to check the
355     PAN and accept broadcast address at some point
356     {_, MacHeader, _} = mac_frame:decode(Frame),
357     case MacHeader#mac_header.dest_addr of
358     ShortAddress -> ok;
359     ExtAddress -> ok;
360     _ -> continue
361     end.
362
363 ack_reply(⟦_, _, #{sys_cfg := #{autoack := 0}} = Regs) ->
364     Regs;
365 ack_reply(Frame, NetworkNode, Regs) ->
366     <<_:2, _ACKREQ:1, _/bitstring>> = Frame,

```

```

360 % io:format("Ack req: ~w ~n ~w", [ACKREQ, Frame]),
361 case Frame of
362 <<?FTYPE_ACK:2, _/bitstring>> ->
363     ok;
364 <<_:2, ?ENABLED:1, _:13, Seqnum:8, _/bitstring>> ->
365     % io:format("Ack requested~n"),
366     Ack = mac_frame:encode_ack(?DISABLED, Seqnum),
367     %io:format("Ack reply from ieee~n"),
368     tx(Ack, #tx_opts{}, NetworkNode, Regs);
369     ->
370     % io:format("No Ack requested~n"),
371     Regs
372 end.

```

```

1 -module(mock_phy).
2 -behaviour(gen_server).
3
4 -export([start_link/2]).
5 -export([start/2]).
6 -export([stop_link/0]).
7
8 -export([transmit/2]).
9 -export([reception_async/0]).
10 -export([reception/0]).
11 -export([reception/1]).
12 -export([disable_rx/0]).
13
14 -export([set_frame_timeout/1]).
15 -export([set_preamble_timeout/1]).
16 -export([disable_preamble_timeout/0]).
17
18 -export([suspend_frame_filtering/0]).
19 -export([resume_frame_filtering/0]).
20
21 -export([read/1]).
22 -export([write/2]).
23
24 -export([rx_ranging_info/0]).
25 -export([signal_power/0]).
26 -export([rx_preamble_repetition/0]).
27 -export([rx_data_rate/0]).
28 -export([prf_value/0]).
29 -export([get_conf/0]).
30
31 %% gen_server callbacks
32 -export([init/1]).
33 -export([handle_call/3]).
34 -export([handle_cast/2]).
35
36 %--- Include -----
37
38 -include("../src/pmod_uwb.hrl").
39
40 %--- Macros -----
41
42 -define(NAME, mock_phy).
43
44 % --- API -----
45
46 start_link(_Connector, Params) ->
47     gen_server:start_link({local, ?NAME}, ?MODULE, Params, []).

```

```

48
49 start(_Connector, Params) ->
50     gen_server:start({local, ?NAME}, ?MODULE, Params, []).
51
52 stop_link() ->
53     gen_server:stop(?NAME).
54
55 transmit(Data, Options) ->
56     gen_server:call(?NAME, {transmit, Data, Options}).
57
58 reception_async() ->
59     Frame = gen_server:call(?NAME, {reception}),
60     Metadata = #{snr => 10.0,
61                 prf => 4,
62                 pre => 16,
63                 data_rate => 1,
64                 rng => ?DISABLED,
65                 rx_stamp => 1,
66                 tx_stamp => 1,
67                 rxtofs => 1,
68                 rxttcki => 1},
69     ieee802154_events:rx_event(Frame, Metadata).
70
71 reception() ->
72     gen_server:call(?NAME, {reception}).
73
74 reception(_) ->
75     gen_server:call(?NAME, {reception}).
76
77 set_frame_timeout(Timeout) ->
78     write(rx_fwto, #{rxfwto => Timeout}),
79     write(sys_cfg, #{rxwtoe => 2#1}). % enable receive wait timeout
80
81 set_preamble_timeout(Timeout) ->
82     write(drx_conf, #{drx_pretoc => Timeout}).
83
84 disable_preamble_timeout() ->
85     write(drx_conf, #{drx_pretoc => 0}).
86
87 suspend_frame_filtering() ->
88     write(sys_cfg, #{fffen => 0}).
89
90 resume_frame_filtering() ->
91     write(sys_cfg, #{fffen => 1}).
92
93 read(Reg) ->
94     gen_server:call(?NAME, {read, Reg}).
95
96 write(Reg, Val) ->
97     gen_server:call(?NAME, {write, Reg, Val}).
98
99 disable_rx() ->
100     gen_server:call(?NAME, {rx_off}).
101
102 %--- API: Getters -----
103 rx_ranging_info() ->
104     #{rng := RNG} = read(rx_finfo),
105     RNG.
106
107 %% @doc Returns the estimated value of the signal power in dBm
108 %% cf. user manual section 4.7.2
109 signal_power() ->

```

```

110     C = channel_impulse_resp_pow() , % Channel impulse response power value (
111         CIR_PWR)
111     A = case prf_value() of
112         16 -> 113.77;
113         64 -> 121.74
114     end, % Constant. For PRF of 16 MHz = 113.77, for PRF of 64MHz = 121.74
115     N = preamble_acc(), % Preamble accumulation count value (RXPACC but might be
116         adjusted)
116     Num = C* math:pow(2, 17),
117     Dem = math:pow(N, 2),
118     Log = math:log10(Num / Dem),
119     10 * Log - A.
120
121 preamble_acc() ->
122     #{rxpacc := RXPACC} = read(rx_finfo),
123     #{rxpacc_nosat := RXPACC_NOSAT} = read(drx_conf),
124     if
125         RXPACC == RXPACC_NOSAT -> RXPACC;
126         true -> RXPACC - 5
127     end.
128
129 channel_impulse_resp_pow() ->
130     #{cir_pwr := CIR_POW} = read(rx_fqual),
131     CIR_POW.
132
133 %% @doc Gives the value of the PRF in MHz
134 -spec prf_value() -> 16 | 64.
135 prf_value() ->
136     #{agc_tune1 := AGC_TUNE1} = read(agc_ctrl),
137     case AGC_TUNE1 of
138         16#8870 -> 16;
139         16#889B -> 64
140     end.
141
142 %% @doc returns the preamble symbols repetition
143 rx_preamble_repetition() ->
144     #{rxpsr := RXPSR} = read(rx_finfo),
145     case RXPSR of
146         0 -> 16;
147         1 -> 64;
148         2 -> 1024;
149         3 -> 4096
150     end.
151
152 %% @doc returns the data rate of the received frame in kbps
153 rx_data_rate() ->
154     #{rxbr := RXBR} = read(rx_finfo),
155     case RXBR of
156         0 -> 110;
157         1 -> 850;
158         3 -> 6800
159     end.
160
161 get_conf() ->
162     gen_server:call(?NAME, {get_conf}).
163
164 %--- gen_server callbacks -----
165 init(_Params) ->
166     {ok, #{regs => pmod_uwb_registers:default(),
167         conf => #phy_cfg{}}}.
168
169 handle_call({transmit, Data, Options}, _From, State) -> {reply, tx(Data, Options),

```



```

    State};
170 handle_call({reception}, _From, State) -> {reply, rx(), State};
171 handle_call({read, Reg}, _From, #{regs := Regs} = State) -> {reply, maps:get(Reg,
    Regs), State};
172 handle_call({write, Reg, Val}, _From, #{regs := Regs} = State) -> {reply, ok,
    State#{regs => pmod_uwb_registers:update_reg(Regs, Reg, Val)}};
173 handle_call({rx_off}, _From, State) -> {reply, ok, State};
174 handle_call({get_conf}, _From, #{conf := Conf} = State) -> {reply, Conf, State};
175 handle_call(_Request, _From, _State) -> error(not_implemented).
176
177 handle_cast(_Request, _State) ->
178     error(not_implemented).
179
180
181 % --- Internal -----
182 tx(_Data, _Options) ->
183     ok.
184
185 rx() ->
186     {14, <<16#6188:16, 0:8, 16#CADE:16, "XR", "XT", "Hello">>}.

```

```

1 -module(mock_top_layer).
2
3 -behaviour(gen_server).
4
5 -include("../src/mac_frame.hrl").
6
7 -include_lib("common_test/include/ct.hrl").
8
9 %% EXPORT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
10 %% API functions
11 -export([start/0]).
12 -export([rx_frame/4]).
13 -export([dump/0]).
14 -export([stop/0]).
15
16 %% gen_server callbacks
17 -export([init/1]).
18 -export([handle_call/3]).
19 -export([handle_cast/2]).
20 -export([handle_info/2]).
21 -export([terminate/2]).
22
23 %% MACROS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
24
25 -define(RCVR_ADDR, <<16#CAFEDCA00000003:64>>).
26 -define(MDL_ADDR, <<16#CAFEDCA00000002:64>>).
27
28 %% API %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
29 start() ->
30     gen_server:start({local, ?MODULE}, ?MODULE, [], []).
31
32 rx_frame(Frame, _, _, _) ->
33     gen_server:cast(?MODULE, {rx, Frame}).
34
35 dump() ->
36     gen_server:call(?MODULE, {dump}).
37
38 stop() ->
39     gen_server:stop(?MODULE).
40

```

```

41 %% gen_server callbacks %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
42 init(_) ->
43     {ok, #{received => #[]}}.
44
45 handle_call({dump}, _From, #{received := Received} = State) ->
46     Ret = lists:sort(fun({_ , MH1, _}, {_ , MH2, _}) ->
47         MH1#mac_header.seqnum < MH2#mac_header.seqnum
48         end, maps:values(Received)),
49     {reply, {ok, Ret}, State#{received => []}};
50 handle_call(_, _, _) ->
51     error(not_implemented).
52
53 handle_cast({rx, {FC, MH, Payload}=Frame}, #{received := Received} = State)
54     when FC#frame_control.frame_type == ?FTYPE_DATA ->
55     Seqnum = MH#mac_header.seqnum,
56     case Received of
57     #{Seqnum := _} ->
58         {noreply, State};
59     _ ->
60         case MH#mac_header.dest_addr of
61         ?MDL_ADDR ->
62             NewMH = MH#mac_header{seqnum = Seqnum + 10,
63                 src_addr = ?MDL_ADDR,
64                 dest_addr = ?RCVR_ADDR},
65             NewFrame = {FC, NewMH, Payload},
66             ieee802154:transmission(NewFrame);
67         _ ->
68             ok
69         end,
70         {noreply, State#{received => maps:put(Seqnum, Frame, Received)}}
71     end;
72 handle_cast({rx, _}, State) ->
73     {noreply, State};
74 handle_cast(_, _) ->
75     error(not_implemented).
76
77 handle_info(_, _) ->
78     error(not_implemented).
79
80 terminate(_, _) ->
81     ok.

```

```

1 -module(network_simulation).
2
3 -behaviour(application).
4
5 -include_lib("common_test/include/ct.hrl").
6
7 -include("../src/mac_frame.hrl").
8
9 %% EXPORTS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
10 %% application callbacks
11 -export([start/2]).
12 -export([stop/1]).
13
14 %% application callbacks %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
15 -spec start(_, Args) -> {ok, pid()} when Args :: #{loss => boolean()}.
16 start(_, Args) ->
17     Loss = maps:get(loss, Args, false),
18     StartData =
19     #{

```

```

20         nodes => sets:new(),
21         exchanges => #{},
22         loss => Loss
23     },
24     LoopPid = spawn(fun() -> loop(ready, StartData) end),
25     register(network_loop, LoopPid),
26     {ok, LoopPid}.
27
28 stop(_) ->
29     network_loop ! {stop},
30     unregister(network_loop).
31
32 %%% Internal %%%
33
34 -spec loop(State, Data) -> ok when
35     State :: ready | blocked,
36     Data :: #{nodes := NodeMap, loss := boolean()},
37     NodeMap :: #{node() => sets:set()}.
38 loop(ready, #{nodes := Nodes} = Data) ->
39     receive
40         {ping, Pid, Node} ->
41             {Pid, Node} ! pong,
42             loop(ready, Data);
43         {register, Name} ->
44             NewNodes = sets:add_element(Name, Nodes),
45             loop(ready, Data#{nodes => NewNodes});
46         {tx, From, Frame} ->
47             do_broadcast(Data, From, Frame);
48         {stop} ->
49             ok
50     end;
51 loop(blocked, #{nodes := Nodes} = Data) ->
52     receive
53         {register, _Name} ->
54             loop(ready, Data);
55         % Once a frame has been blocked => pass through
56         {tx, From, Frame} ->
57             broadcast(sets:to_list(Nodes), From, Frame),
58             loop(blocked, Data);
59         {stop} ->
60             ok;
61         OtherEvent ->
62             error(wrong_event_in_blocked, OtherEvent)
63     end.
64
65 do_broadcast(#{loss := false, nodes := Nodes} = Data, From, Frame) ->
66     broadcast(sets:to_list(Nodes), From, Frame),
67     loop(ready, Data);
68 do_broadcast(#{loss := true} = Data, From, Frame) ->
69     {_, RawFrame} = Frame,
70     #{nodes := Nodes, exchanges := Exchanges} = Data,
71     {_, MH, _} = mac_frame:decode(RawFrame),
72     #mac_header{dest_addr = DestAddr} = MH,
73     Key = {From, DestAddr},
74     case maps:get(Key, Exchanges, {0, 0}) of
75         {MemorySeen, MemorySeen} ->
76             NewExch = maps:put(Key, {0, MemorySeen + 1}, Exchanges),
77             loop(blocked, Data#{exchanges => NewExch});
78         {RndSeen, MemorySeen} ->
79             broadcast(sets:to_list(Nodes), From, Frame),
80             NewExch = maps:put(Key, {RndSeen + 1, MemorySeen}, Exchanges),
81             loop(ready, Data#{exchanges => NewExch})

```

```

82     end.
83
84 broadcast([], _, _) ->
85     ok;
86 broadcast([From | T], From, Frame) ->
87     broadcast(T, From, Frame);
88 broadcast([Node | T], From, Frame) ->
89     {mock_phy_network, Node} ! {frame, Frame},
90     broadcast(T, From, Frame).

```

```

1 -module(pmod_uwb_registers).
2
3 -export([default/0]).
4 -export([update_reg/3]).
5 -export([get_value/2]).
6
7 -spec default() -> map().
8 default() ->
9     #{eui => #{eui => <<16#FFFFFFFF00000000:64>>}, % 0x01
10     panadr => #{pan_id => <<16#FFFFFFFF:16>>, short_addr => <<16#FFFFFFFF:16>>},
11         % 0x03
12     sys_cfg => #{aackpend => 0, % 0x04
13                 autoack => 0,
14                 rxautr => 0,
15                 rxwtoe => 0,
16                 rxm110k => 0,
17                 dis_stxp => 0,
18                 phr_mode => 0,
19                 fcs_init2f => 0,
20                 dis_rsde => 0,
21                 dis_phe => 0,
22                 dis_drxb => 1,
23                 dis_fce => 0,
24                 spi_edge => 0,
25                 hirq_pol => 1,
26                 ffa5 => 0,
27                 ffa4 => 0,
28                 ffar => 0,
29                 ffam => 0,
30                 ffaa => 0,
31                 ffad => 0,
32                 ffab => 0,
33                 ffbc => 0,
34                 ffen => 0},
35     tx_fctrl => #{txboffs => 0, % 0x08
36                 pe => 1,
37                 txpsr => 1,
38                 txprf => 1,
39                 tr => 0,
40                 txbr => 2,
41                 r => 0,
42                 tfile => 0,
43                 tflen => 12},
44     rx_fwto => #{rxfwto => 0}, % 0x0C
45     sys_ctrl => #{sfcst => 0, % 0x0D
46                 txstrt => 0,
47                 txdlys => 0,
48                 trxoff => 0,
49                 wait4resp => 0,
50                 rxenab => 0,
51                 rxdlye => 0,

```

```

51         hrbpbt => 0},
52     sys_status => #{irqs => 0, % 0x0F
53         cplck => 0,
54         esyncr => 0,
55         aat => 0,
56         txfrb => 0,
57         txprs => 0,
58         pxphs => 0,
59         txfrs => 0,
60         rxprd => 0,
61         rxsfdd => 0,
62         ldedone => 0,
63         rxphd => 0,
64         rxphe => 0,
65         rxdfc => 0,
66         rxfcg => 0,
67         rxfce => 0,
68         rxrfs1 => 0,
69         rxrfto => 0,
70         ldeerr => 0,
71         rxovrr => 0,
72         rxpto => 0,
73         gpioirq => 0,
74         slp2init => 0,
75         rfp11_11 => 0,
76         clkpll_11 => 0,
77         rxsfdto => 0,
78         hpdwarn => 0,
79         txberr => 0,
80         affrej => 0,
81         hsrbp => 0,
82         icrbp => 0,
83         rxrscs => 0,
84         rxprej => 0,
85         txpute => 0},
86     rx_finfo => #{rxpacc => 1025, % 0x10
87         rxpsr => 0,
88         rxprfr => 0,
89         rng => 0,
90         rxbr => 0,
91         rxfle => 0,
92         rxflen => 0},
93     rx_buffer => #{rx_buffer => <<>>, % 0x11
94     rx_fqual => #{fp_ampl2 => 0, % 0x12
95         std_noise => 0,
96         cir_pwr => 1,
97         pp_ampl3 => 0},
98     rx_ttcki => #{rxttcki => 0}, % 0x13
99     rx_ttcko => #{rmspdel => 0, % 0x14
100         rxtofs => 0,
101         rcphase => 0},
102     rx_time => #{rx_stamp => 0, % 0x15
103         fp_index => 0,
104         fp_ampl1 => 0,
105         rx_rawst => 0},
106     tx_time => #{tx_stamp => 0, % 0x17
107         tx_rawst => 0},
108     rx_sniff => #{sniff_offt => 0, sniff_ont => 0}, % 0x1D
109     agc_ctrl => #{agc_ctrl1 => #{dis_am => 1},
110         agc_tune1 => 16#8870,
111         agc_tune2 => 16#2502A907,
112         agc_tune3 => 16#0035,

```

```

113         agc_stat1 => #{edv2 => 0,
114                    edg1 => 1}},
115     % DRX_CONF isn't complete yet
116     drx_conf => #{drx_pretoc => 0, % 0x27
117                rxpacc_nosat => 0},
118     % PMSC isn't complete yet
119     pmsc => #{pmsc_ctrl0 => #{}, % 0x36
120             pmsc_ctrl1 => #{arx2init => 0}}
121     }.
122
123 -spec update_reg(Regs::map(), Reg::atom(), NewVal::atom()|map() -> map().
124 update_reg(Regs, Reg, NewVal) ->
125     OldVal = maps:get(Reg, Regs),
126     if is_map(OldVal) -> maps:put(Reg, maps:merge(OldVal, NewVal), Regs);
127     true -> maps:put(Reg, NewVal, Regs) end.
128
129 -spec get_value(Regs::map(), Reg::atom()) -> atom()|map().
130 get_value(Regs, Reg) ->
131     maps:get(Reg, Regs).

```

A.13 Mac layer code

```

1 -module(ieee802154).
2 -behaviour(gen_server).
3
4 %%% @headerfile "ieee802154.hrl"
5
6 % API
7 -export([start_link/1]).
8 -export([start/1]).
9 -export([stop_link/0]).
10 -export([stop/0]).
11
12 -export([transmission/1]).
13 -export([transmission/2]).
14 -export([reception/0]).
15
16 -export([rx_on/0]).
17 -export([rx_off/0]).
18
19 -export([get_pib_attribute/1]).
20 -export([set_pib_attribute/2]).
21
22 -export([reset/1]).
23
24 % gen_server callbacks
25 -export([init/1]).
26 -export([terminate/2]).
27 -export([code_change/4]).
28 -export([handle_call/3]).
29 -export([handle_cast/2]).
30
31
32 % Includes
33 -include("ieee802154.hrl").
34 -include("ieee802154_pib.hrl").
35 -include("mac_frame.hrl").

```

```

36
37 %--- Types -----
38
39 -type state() :: #{phy_layer := module(),
40                 duty_cycle := gen_duty_cycle:state(),
41                 pib := pib_state(),
42                 _:=_}.
43
44 %--- API -----
45
46 %% @doc Starts the IEEE 812.15.4 stack and creates a link
47 %%
48 %% '''
49 %% The following code will start the stack with the default parameters
50 %% 1> ieee802154:start_link(#ieee_parameters{}).
51 %%
52 %% Using a custom callback function
53 %% 2> ieee802154:start_link(#ieee_parameters{input_callback = fun callback/4}).
54 %%
55 %% Using a custom phy module
56 %% 3> ieee802154:start_link(#ieee_parameters{phy_layer = mock_phy_network}).
57 %% '''
58 %%
59 %% @param Params: A map containing the parameters of the IEEE stack
60 %%
61 %% @end
62 -spec start_link(Params) -> {ok, pid()} | {error, any()} when
63     Params :: ieee_parameters().
64 start_link(Params) ->
65     gen_server:start_link({local, ?MODULE}, ?MODULE, Params, []).
66
67 %% @doc Same as start_link/1 but no link is created
68 %% @end
69 -spec start(Params) -> {ok, pid()} | {error, any()} when
70     Params :: ieee_parameters().
71 start(Params) ->
72     gen_server:start({local, ?MODULE}, ?MODULE, Params, []).
73
74 stop_link() ->
75     gen_server:stop(?MODULE).
76
77 stop() -> gen_server:stop(?MODULE).
78
79 %% @doc
80 %% @equiv transmission(Frame, 0)
81 %% @end
82 -spec transmission(Frame) -> Result when
83     Frame      :: frame(),
84     Result     :: {ok, Ranging} | {error, Error},
85     Ranging    :: ranging_informations(),
86     Error      :: tx_error().
87 transmission(Frame) ->
88     transmission(Frame, ?NON_RANGING).
89
90 %% @doc Performs a transmission on the defined IEEE 802.15.4 stack
91 %% When ranging has been activated for the frame, the second element of the
92 %% tuple contains different values that can be used for ranging operations
93 %% For more informations please consult the IEEE 802.15.4 standard.
94 %% Note that the variable 'Timestamp' is omitted because its value is the same
95 %% as 'Ranging counter start'
96 %%
97 %% When Ranging isn't activated, the 2nd element of the tuple shall be ignored

```

```

98 %% '''
99 %% Ranging not activated for transmission
100 %% 1> ieee802154:transmission(Frame, ?NON_RANGING).
101 %%
102 %% Activate ranging for the transmission
103 %% 2> ieee802154:transmission(Frame, ?ALL_RANGING).
104 %% '''
105 %% @end
106 -spec transmission(Frame, Ranging) -> Result when
107     Frame      :: frame(),
108     Ranging    :: ranging_tx(),
109     Result     :: {ok, RangingInfos} | {error, Error},
110     RangingInfos :: ranging_informations(),
111     Error      :: tx_error().
112 transmission(Frame, Ranging) ->
113     {FH, _, _} = Frame,
114     case FH of
115         #frame_control{dest_addr_mode = ?NONE, src_addr_mode = ?NONE} ->
116             {error, invalid_address};
117         _ ->
118             gen_server:call(?MODULE,
119                             {tx, Frame, Ranging},
120                             infinity)
121     end.
122
123 %% @doc Performs a reception on the IEEE 802.15.4 stack
124 %% @deprecated This function will be deprecated
125 %% @end
126 -spec reception() -> Result when
127     Result :: {ok, frame()} | {error, atom()}.
128 reception() ->
129     gen_server:call(?MODULE, {rx}, infinity).
130
131 %% @doc Turns on the continuous reception
132 %% Ranging is by default switched on
133 %% @end
134 -spec rx_on() -> Result when
135     Result :: ok | {error, atom()}.
136 rx_on() ->
137     gen_server:call(?MODULE, {rx_on}).
138
139 %% @doc Turns off the continuous reception
140 %% @end
141 rx_off() ->
142     gen_server:call(?MODULE, {rx_off}).
143
144 %% @doc Get the value of a PIB attribute
145 %% @end
146 -spec get_pib_attribute(Attribute) -> Value when
147     Attribute :: pib_attribute(),
148     Value     :: term().
149 get_pib_attribute(Attribute) ->
150     gen_server:call(?MODULE, {get, Attribute}).
151
152
153 %% @doc Set the value of a PIB attribute
154 %% @end
155 -spec set_pib_attribute(Attribute, Value) -> ok when
156     Attribute :: pib_attribute(),
157     Value     :: term().
158 set_pib_attribute(Attribute, Value) ->
159     gen_server:call(?MODULE, {set, Attribute, Value}).

```



```

160
161 -spec reset(SetDefaultPIB) -> Result when
162     SetDefaultPIB :: boolean(),
163     Result :: ok.
164 reset(SetDefaultPIB) ->
165     gen_server:call(?MODULE, {reset, SetDefaultPIB}).
166
167 %--- gen_statem callbacks -----
168
169 -spec init(Params) -> {ok, State} when
170     Params :: ieee_parameters(),
171     State :: state().
172 init(Params) ->
173     {ok, _GenEvent} = gen_event:start_link({local, ?GEN_EVENT}),
174     PhyMod = Params#ieee_parameters.phy_layer,
175     InputCallback = Params#ieee_parameters.input_callback,
176     write_default_conf(PhyMod),
177
178     ok = ieee802154_events:start(#{input_callback => InputCallback}),
179
180     DutyCycleState = gen_duty_cycle:start(Params#ieee_parameters.duty_cycle,
181                                         PhyMod),
182
183     Data = #{phy_layer => PhyMod,
184             duty_cycle => DutyCycleState,
185             pib => ieee802154_pib:init(PhyMod),
186             ranging => ?DISABLED},
187     {ok, Data}.
188
189 -spec terminate(Reason, State) -> ok when
190     Reason :: term(),
191     State :: state().
192 terminate(Reason, #{duty_cycle := GenDutyCycleState}) ->
193     ieee802154_events:stop(),
194     gen_event:stop(?GEN_EVENT),
195     gen_duty_cycle:stop(GenDutyCycleState, Reason).
196
197 code_change(_, _, _, _) ->
198     error(not_implemented).
199
200 -spec handle_call(_, _, State) -> Result when
201     State :: state(),
202     Result :: {reply, term(), State}.
203 handle_call({rx_on}, _From, State) ->
204     #{duty_cycle := DCState} = State,
205     case gen_duty_cycle:turn_on(DCState) of
206     {ok, NewDutyCycleState} ->
207         {reply, ok, State#{duty_cycle => NewDutyCycleState}};
208     {error, NewDutyCycleState, Error} ->
209         {reply, {error, Error}, State#{duty_cycle => NewDutyCycleState}}
210     end;
211 handle_call({rx_off}, _From, #{duty_cycle := DCState} = State) ->
212     NewDCState = gen_duty_cycle:turn_off(DCState),
213     {reply, ok, State#{duty_cycle => NewDCState}};
214 handle_call({tx, Frame, Ranging}, _From, State) ->
215     #{duty_cycle := DCState, pib := Pib} = State,
216     {FrameControl, MacHeader, Payload} = Frame,
217     EncFrame = mac_frame:encode(FrameControl, MacHeader, Payload),
218     case gen_duty_cycle:tx_request(DCState, EncFrame, Pib, Ranging) of
219     {ok, NewDCState, RangingInfos} ->
220         timer:sleep(100), % FIXME: IFS
221         {reply, {ok, RangingInfos}, State#{duty_cycle => NewDCState}};

```

```

222     {error, NewDCState, Error} ->
223         {reply, {error, Error}, State#{duty_cycle => NewDCState}}
224     end;
225 handle_call({get, Attribute}, _From, State) ->
226     #{pib := Pib} = State,
227     case ieee802154_pib:get(Pib, Attribute) of
228     {error, Error} ->
229         {reply, {error, Error}, State};
230     Value ->
231         {reply, Value, State}
232     end;
233 handle_call({set, Attribute, Value}, _From, State) ->
234     #{pib := Pib} = State,
235     case ieee802154_pib:set(Pib, Attribute, Value) of
236     {ok, NewPib} ->
237         {reply, ok, State#{pib => NewPib}};
238     {error, NewPib, Error} ->
239         {reply, {error, Error}, State#{pib => NewPib}}
240     end;
241 handle_call({reset, SetDefaultPIB}, _From, State) ->
242     #{phy_layer := PhyMod, pib := Pib, duty_cycle := DCState} = State,
243     NewState = case SetDefaultPIB of
244         true ->
245             PhyMod:write(panadr, #{pan_id => <<16#FFFF:16>>,
246                             short_addr => <<16#FFFF:16>>}),
247             State#{pib => ieee802154_pib:reset(Pib)};
248         _ ->
249             State
250     end,
251     NewDCState = gen_duty_cycle:turn_off(DCState),
252     {reply, ok, NewState#{duty_cycle => NewDCState, ranging => ?DISABLED}};
253 handle_call(_Request, _From, _State) ->
254     error(call_not_recognized).
255
256 handle_cast(_, _) ->
257     error(not_implemented).
258
259 %--- Internal -----
260 -spec write_default_conf(PhyMod :: module()) -> ok.
261 write_default_conf(PhyMod) ->
262     PhyMod:write(rx_fwto, #{rxfwto => ?MACACKWAITDURATION}),
263     PhyMod:write(sys_cfg, #{ffab => 1,
264                             ffad => 1,
265                             ffaa => 1,
266                             ffam => 1,
267                             ffen => 1,
268                             autoack => 1,
269                             rxwtoe => 1}).

```

```

1 %--- Macros -----
2
3 %--- MCPS-DATA.indication Parameters
4
5 % Ranging Received values:
6 -define(NO_RANGING_REQUESTED, 0).
7 -define(RANGING_REQUESTED_BUT_NOT_SUPPORTED, 1).
8 -define(RANGING_ACTIVE, 2).
9
10 % Ranging Transmission values:
11 -define(NON_RANGING, 0).
12 -define(ALL_RANGING, 1).

```

```

13 % PHY_HEADER_ONLY => Not supported in our case
14
15 % CSMA constants
16 -define(MACMAXFRAMERETRIES, 5).
17 -define(MACACKWAITDURATION, 4000). % works with 2000 s but calculations give me
18 % -define(MACACKWAITDURATION, 2000). % works with 2000 s but calculations give
19 % me 4081 s
20 -define(GEN_EVENT, gen_event).
21
22 %--- Types -----
23 %--- Record types
24 -record(ieee_parameters, {duty_cycle = duty_cycle_non_beacon :: module(),
25                          phy_layer = pmod_uwb :: module(),
26                          input_callback = fun(_, _, _, _) -> ok end ::
27                          input_callback()}).
28
29 -record(ranging_informations, {ranging_received = ?NO_RANGING_REQUESTED ::
30                               ranging_received() | boolean(),
31                               ranging_counter_start = 0 :: integer
32                               (),
33                               ranging_counter_stop = 0 :: integer
34                               (),
35                               ranging_tracking_interval = 0 :: integer
36                               (),
37                               ranging_offset = 0 :: integer
38                               (),
39                               ranging_FOM = <<16#00:8>> ::
40                               bitstring()}).
41
42 -type ranging_informations() :: #ranging_informations{}.
43
44 % For now security isn't enabled
45 -record(security, {security_level = 0 :: integer(),
46                  key_id_mode = 0 :: integer(),
47                  key_source = <<16#00:8>> :: bitstring()}).
48
49 -type security() :: #security{}.
50
51 %--- IEEE 802.15.4 parameter types
52 -export_type([ieee_parameters/0, ranging_informations/0, security/0,
53              input_callback/0, ranging_tx/0, tx_error/0]).
54
55 -type ranging_received() :: ?NO_RANGING_REQUESTED | ?
56   RANGING_REQUESTED_BUT_NOT_SUPPORTED | ?RANGING_ACTIVE.
57 -type ranging_tx() :: ?NON_RANGING | ?ALL_RANGING. % PHY_HEADER_ONLY no used in
58   our case
59
60 % *** indicates unusefull parameters for higher layers for now
61 -type input_callback() :: fun((Frame :: mac_frame:frame(),
62                               LQI :: integer(),
63                               % UWBPRF :: gen_mac_layer:uwb_PRF()
64                               ),
65                               Security :: security(),
66                               % UWBpreambleRepetitions :: pmod_uwb:
67                               uwb_preamble_symbol_repetition(),
68                               % DataRate :: pmod_uwb:data_rate(),
69                               Ranging :: ranging_informations())
70   -> ok).

```

```

61 -type ieee_parameters() :: #ieee_parameters{}.
62
63 -type tx_error() :: invalid_address | invalid_gts | transaction_overflow |
    transaction_expired | no_ack | frame_too_long | channel_access_failure.

```

```

1 -module(ieee802154_events).
2
3 -behaviour(gen_event).
4
5 %--- Exports -----
6
7 -export([start/1]).
8 -export([stop/0]).
9 -export([rx_event/2]).
10
11 % gen_event callbacks
12 -export([init/1]).
13 -export([handle_event/2]).
14 -export([handle_call/2]).
15 -export([handle_info/2]).
16 -export([terminate/2]).
17
18 %--- Includes -----
19
20 -include("ieee802154.hrl").
21 -include("pmod_uwb.hrl").
22
23 %--- Records -----
24
25 -record(state, {input_callback :: ieee802154:input_callback()}).
26
27 %--- Types -----
28
29 -type state() :: #state{}.
30
31 %--- API -----
32 % TODO:
33 % * Notify an event
34 % * Subscribe to an event => Wait for an event to happen
35 % * Add a callback for an event ? => Are these events ?
36 -spec start(Args :: map()) -> ok.
37 start(Args) ->
38     gen_event:add_handler(?GEN_EVENT, ?MODULE, Args).
39
40 stop() ->
41     gen_event:delete_handler(?GEN_EVENT, ?MODULE, []).
42
43 % @doc triggers a rx event
44 % -spec rx_event(Frame, Metadata) -> ok when
45 %     Frame :: {integer(), bitstring()},
46 %     Metadata :: #{snr := float(),
47 %                 prf := uwb_PRF(),
48 %                 pre := uwb_preamble_symbol_repetition(),
49 %                 data_rate := data_rate(),
50 %                 rng := flag(),
51 %                 rx_stamp := integer(),
52 %                 tx_stamp := integer(),
53 %                 rxtofs := integer(),
54 %                 rxttcki := integer()}.
55 rx_event(_, Frame, Metadata) ->
56     gen_event:notify(?GEN_EVENT, {rx, Frame, Metadata}).

```

```

57
58 %--- gen_event callbacks -----
59 -spec init(InitArgs :: map()) ->
60   {ok, State :: state()}.
61 init(State) ->
62   #{input_callback := InputCallback} = State,
63   {ok, #state{input_callback = InputCallback}}.
64
65 handle_event({rx, Frame, Metadata}, State) ->
66   #state{input_callback = InputCallback} = State,
67   DecodedFrame = mac_frame:decode(Frame),
68   #{snr := Snr,
69     rng := Rng,
70     rx_stamp := RxStamp,
71     tx_stamp := TxStamp,
72     rxtofs := Rxtofs,
73     rxttcki := Rxttcki} = Metadata,
74   RngInfo = rng_infos(Rng, RxStamp, TxStamp, Rxtofs, Rxttcki),
75   InputCallback(DecodedFrame, Snr, #security{}, RngInfo),
76   {ok, State};
77 handle_event(_Event, State) ->
78   {ok, State}.
79
80 handle_call(_, State) ->
81   % TODO user should be able to register a callback
82   {ok, ok, State}.
83
84 handle_info(_, State) ->
85   % TODO: nothing here
86   {ok, State}.
87
88 terminate(_, _) ->
89   ok.
90
91 %--- Internal -----
92
93 rng_infos(?ENABLED, RxStamp, TxStamp, Rxtofs, Rxttcki) ->
94   #ranging_informations{
95     ranging_received = ?RANGING_ACTIVE,
96     ranging_counter_start = RxStamp,
97     ranging_counter_stop = TxStamp,
98     ranging_tracking_interval = Rxttcki,
99     ranging_offset = Rxtofs,
100    ranging_FOM = <<0:8>>
101   };
102 rng_infos(?DISABLED, _, _, _, _) ->
103   #ranging_informations{ranging_received = ?NO_RANGING_REQUESTED}.

```

```

1 -module(ieee802154_pib).
2
3 -export([init/1]).
4 -export([get/2]).
5 -export([set/3]).
6 -export([reset/1]).
7
8 -include("ieee802154_pib.hrl").
9
10 %--- API -----
11
12 init(PhyMod) ->
13   {PhyMod, default_attributes()}.

```

```

14
15 -spec get(State, Attribute) -> Result when
16     State      :: pib_state(),
17     Attribute  :: pib_attribute(),
18     Result     :: Value | {error, unsupported_attribute},
19     Value      :: term().
20 get({_, Attributes}, Attribute) when is_map_key(Attribute, Attributes) ->
21     maps:get(Attribute, Attributes);
22 get(_, _) ->
23     {error, unsupported_attribute}.
24
25 -spec set(State, Attribute, Value) -> Results when
26     State      :: {PhyMod, Attributes},
27     PhyMod     :: module(),
28     Attributes :: pib_attributes(),
29     Attribute  :: pib_attribute(),
30     Value      :: term(),
31     Results    :: {ok, NewState} | {error, NewState, Error},
32     NewState   :: pib_state(),
33     Error      :: pib_set_error().
34 set({PhyMod, Attributes}, mac_extended_address, Value) ->
35     PhyMod:write(eui, #{eui => Value}), % TODO check the range/type/value given
36     {ok, {PhyMod, Attributes#{mac_extended_address => Value}}};
37 set({PhyMod, Attributes}, mac_short_address, Value) ->
38     PhyMod:write(panadr, #{short_addr => Value}),
39     {ok, {PhyMod, Attributes#{mac_short_address => Value}}};
40 set({PhyMod, Attributes}, mac_pan_id, Value) ->
41     PhyMod:write(panadr, #{pan_id => Value}),
42     {ok, {PhyMod, Attributes#{mac_pan_id => Value}}};
43 set({PhyMod, Attributes}, Attribute, Value)
44     when is_map_key(Attribute, Attributes) ->
45     NewAttributes = maps:update(Attribute, Value, Attributes),
46     {ok, {PhyMod, NewAttributes}};
47 set(State, _, _) ->
48     % TODO detect if PIB is a read only attribute
49     {error, State, unsupported_attribute}.
50
51 reset({PhyMod, _}) ->
52     {PhyMod, default_attributes()}.
53
54 %--- Internal -----
55 default_attributes() ->
56     #{
57         cw0 => 2, % cf. p.22 standard
58         mac_extended_address => <<16#FFFFFFFF00000000:64>>,
59         % mac_max_BE => 8,
60         mac_max_BE => 5,
61         mac_max_csma_backoffs => 4,
62         % mac_min_BE => 5,
63         mac_min_BE => 3,
64         mac_pan_id => <<16#FFFF:16>>,
65         mac_short_address => <<16#FFFF:16>>
66     }.

```

```

1 -module(ieee802154_pib).
2
3 -export([init/1]).
4 -export([get/2]).
5 -export([set/3]).
6 -export([reset/1]).
7

```

```

8 -include("ieee802154_pib.hrl").
9
10 %--- API -----
11
12 init(PhyMod) ->
13     {PhyMod, default_attributes()}.
14
15 -spec get(State, Attribute) -> Result when
16     State      :: pib_state(),
17     Attribute  :: pib_attribute(),
18     Result     :: Value | {error, unsupported_attribute},
19     Value      :: term().
20 get(_, Attributes, Attribute) when is_map_key(Attribute, Attributes) ->
21     maps:get(Attribute, Attributes);
22 get(_, _) ->
23     {error, unsupported_attribute}.
24
25 -spec set(State, Attribute, Value) -> Results when
26     State      :: {PhyMod, Attributes},
27     PhyMod     :: module(),
28     Attributes :: pib_attributes(),
29     Attribute  :: pib_attribute(),
30     Value      :: term(),
31     Results    :: {ok, NewState} | {error, NewState, Error},
32     NewState   :: pib_state(),
33     Error      :: pib_set_error().
34 set({PhyMod, Attributes}, mac_extended_address, Value) ->
35     PhyMod:write(eui, #{eui => Value}), % TODO check the range/type/value given
36     {ok, {PhyMod, Attributes#{mac_extended_address => Value}}};
37 set({PhyMod, Attributes}, mac_short_address, Value) ->
38     PhyMod:write(panadr, #{short_addr => Value}),
39     {ok, {PhyMod, Attributes#{mac_short_address => Value}}};
40 set({PhyMod, Attributes}, mac_pan_id, Value) ->
41     PhyMod:write(panadr, #{pan_id => Value}),
42     {ok, {PhyMod, Attributes#{mac_pan_id => Value}}};
43 set({PhyMod, Attributes}, Attribute, Value)
44     when is_map_key(Attribute, Attributes) ->
45     NewAttributes = maps:update(Attribute, Value, Attributes),
46     {ok, {PhyMod, NewAttributes}};
47 set(State, _, _) ->
48     % TODO detect if PIB is a read only attribute
49     {error, State, unsupported_attribute}.
50
51 reset({PhyMod, _}) ->
52     {PhyMod, default_attributes()}.
53
54 %--- Internal -----
55 default_attributes() ->
56     #{
57         cw0 => 2, % cf. p.22 standard
58         mac_extended_address => <<16#FFFFFFFF00000000:64>>,
59         % mac_max_BE => 8,
60         mac_max_BE => 5,
61         mac_max_csma_backoffs => 4,
62         % mac_min_BE => 5,
63         mac_min_BE => 3,
64         mac_pan_id => <<16#FFFF:16>>,
65         mac_short_address => <<16#FFFF:16>>
66     }.

```

```

1 -module(ieee802154_utils).

```

```

2
3 %--- Export -----
4
5 -export([pkt_duration/2]).
6 -export([t_dsym/1]).
7 -export([symbols_to_usec/2]).
8
9 %--- Include -----
10
11 -include("pmod_uwb.hrl").
12
13 %--- Macros -----
14
15 -define(N_PHR, 19).
16 -define(T_D_SYM_1M, 1025.64).
17
18 %--- API -----
19
20 % @doc get the packet duration in ns
21 pkt_duration(PcktSize, Conf) ->
22     #phy_cfg{prf = PRF, psr = PSR, sfd = SFD} = Conf,
23     TShr = t_psym(PRF) * (PSR + SFD),
24     TPhr = ?N_PHR * ?T_D_SYM_1M,
25     TDsym = t_dsym(Conf),
26     TPdsu = TDsym * PcktSize * 8,
27     TShr + TPhr + TPdsu.
28
29 t_psym(16) ->
30     993.6;
31 t_psym(64) ->
32     1017.6;
33 t_psym(PRF) ->
34     error({non_supported_prf, PRF}).
35
36 % t_dsym according to values of table 99 Std. IEEE.802.15.4
37 % Over all the possible values supported for the PRF and the channels
38 % only the bit rate determines T_dsym
39 t_dsym(#phy_cfg{data_rate = ?DATA_RATE_11KHZ}) ->
40     8205.13;
41 t_dsym(#phy_cfg{data_rate = ?DATA_RATE_84KHZ}) ->
42     1025.64;
43 t_dsym(#phy_cfg{data_rate = ?DATA_RATE_6MHZ}) ->
44     128.21;
45 t_dsym(BitRate) ->
46     error({non_supported_bit_rate, BitRate}).
47
48 % @doc Converts a value in symbols to usec
49 % It uses t_psym to perform the conversion (defined in table 99)
50 % @end
51 symbols_to_usec(Symbols, #phy_cfg{prf = PRF}) ->
52     Symbols * t_psym(PRF) / 1000.

```

```

1 -module(mac_frame).
2
3 -include("mac_frame.hrl").
4
5 -export([encode/2]).
6 -export([encode/3]).
7 -export([encode_ack/2]).
8 -export([decode/1]).
9

```



```

10 %-----
11 % @doc builds a mac frame without a payload
12 % @equiv encode(FrameControl, MacHeader, <<>>)
13 % @end
14 %-----
15 -spec encode(FrameControl :: #frame_control{}, MacHeader :: #mac_header{}) ->
    bitstring().
16 encode(FrameControl, MacHeader) ->
17     encode(FrameControl, MacHeader, <<>>).
18
19 %-----
20 % @doc builds a mac frame
21 % @returns a MAC frame ready to be transmitted in a bitstring (not including the
    CRC automatically added by the DW1000)
22 % @end
23 %-----
24 -spec encode(
25     FrameControl :: frame_control(),
26     MacHeader :: mac_header(),
27     Payload :: bitstring()
28 ) ->
    bitstring().
29 encode(FrameControl, MacHeader, Payload) ->
30     Header = build_mac_header(FrameControl, MacHeader),
31     <<Header/bitstring, Payload/bitstring>>.
32
33 %-----
34 % @doc Builds an ACK frame
35 % @returns a MAC frame ready to be transmitted in a bitstring (not including the
    CRC automatically added by the DW1000)
36 % @end
37 %-----
38 %-----
39 encode_ack(FramePending, Seqnum) ->
40     FC = build_frame_control(#frame_control{
41         frame_type = ?FTYPE_ACK,
42         frame_pending = FramePending,
43         dest_addr_mode = ?NONE,
44         src_addr_mode = ?NONE
45     }),
46     <<FC/bitstring, Seqnum:8>>.
47
48 %-----
49 % @doc builds a mac header based on the FrameControl and the MacHeader structures
    given in the args.
50 % <b> The MAC header doesn't support security fields yet </b>
51 % @returns the MAC header in a bitstring
52 % @end
53 %-----
54 -spec build_mac_header(FrameControl, MacHeader) -> binary() when
55     FrameControl :: frame_control(),
56     MacHeader :: mac_header().
57 build_mac_header(FrameControl, MacHeader) ->
58     FC = build_frame_control(FrameControl),
59
60     DestPan = reverse_byte_order(MacHeader#mac_header.dest_pan),
61     DestAddr = reverse_byte_order(MacHeader#mac_header.dest_addr),
62     DestAddrFields =
63         case FrameControl#frame_control.dest_addr_mode of
64             ?NONE ->
65                 <<>>;
66             - ->
67                 <<DestPan/bitstring, DestAddr/bitstring>>

```

```

68     end,
69
70     SrcPan = reverse_byte_order(MacHeader#mac_header.src_pan),
71     SrcAddr = reverse_byte_order(MacHeader#mac_header.src_addr),
72     SrcAddrFields =
73         case
74         {
75             FrameControl#frame_control.src_addr_mode,
76             FrameControl#frame_control.pan_id_compr,
77             FrameControl#frame_control.dest_addr_mode
78         }
79     of
80     {?NONE, _, _} ->
81         <<>>;
82     {_, ?DISABLED, _} ->
83         <<SrcPan/bitstring,
84             % if no compression is applied on PANID and SRC addr is
85             present
86             SrcAddr/bitstring>>;
87     {_, ?ENABLED, ?NONE} ->
88         <<SrcPan/bitstring,
89             % if there is a compression of the PANID but the dest addr isn
90             't present
91             SrcAddr/bitstring>>;
92     {_, ?ENABLED, _} ->
93         % if there is a compression of the PANID and the dest addr is
94         present
95         <<SrcAddr/bitstring>>
96     end,
97     <<FC/bitstring, (MacHeader#mac_header.seqnum):8, DestAddrFields/bitstring,
98     SrcAddrFields/bitstring>>.
99
100 %-----
101 % @doc decodes the MAC frame given in the arguments
102 % @return A tuple containing the decoded frame control, the decoded mac header and
103 % the payload
104 % @end
105 %-----
106 -spec decode(Data) -> {FrameControl, MacHeader, Payload} when
107     Data :: binary(),
108     FrameControl :: frame_control(),
109     MacHeader :: mac_header(),
110     Payload :: bitstring().
111 decode(Data) ->
112     <<FC:16/bitstring, Seqnum:8, Rest/bitstring>> = Data,
113     FrameControl = decode_frame_control(FC),
114     decode_rest(FrameControl, Seqnum, Rest).
115
116 %-----
117 % @private
118 % @doc Decodes the remaining sequence of bit present in the payload after the
119 % seqnum
120 % @end
121 %-----
122 -spec decode_rest(
123     FrameControl :: frame_control(),
124     Seqnum :: integer(),
125     Rest :: binary()
126 ) ->
127     {FrameControl :: frame_control(), MacHeader :: mac_header(), Payload :: binary
128     ()}.
129 decode_rest(

```

```

123     #frame_control{frame_type = ?FTYPE_ACK} = FrameControl,
124     Seqnum,
125     % Might cause an issue if piggybacking is used (allowed in IEEE 802.15.4?)
126     _Rest
127 ) ->
128     {FrameControl, #mac_header{seqnum = Seqnum}, <<>>};
129 decode_rest(FrameControl, Seqnum, Rest) ->
130     {DestPAN, DestAddr, SrcPAN, SrcAddr, Payload} =
131         decode_mac_header(
132             FrameControl#frame_control.dest_addr_mode,
133             FrameControl#frame_control.src_addr_mode,
134             FrameControl#frame_control.pan_id_compr,
135             Rest
136         ),
137     MacHeader =
138         #mac_header{
139             seqnum = Seqnum,
140             dest_pan = reverse_byte_order(DestPAN),
141             dest_addr = reverse_byte_order(DestAddr),
142             src_pan = reverse_byte_order(SrcPAN),
143             src_addr = reverse_byte_order(SrcAddr)
144         },
145     {FrameControl, MacHeader, Payload}.
146
147 % Note extended addresses and PAN ID are used in the case of inter-PAN
148 % communication
149 % In inter PAN communication, it can be omitted but it's not mandatory
150 -spec decode_mac_header(DestAddrMode, SrcAddrMode, PanIdCompr, Bits) ->
151     {DestPAN, DestAddr, SrcPAN, SrcAddr, Payload}
152 when
153     DestAddrMode :: flag(),
154     SrcAddrMode :: flag(),
155     PanIdCompr :: flag(),
156     Bits :: bitstring(),
157     DestPAN :: binary(),
158     DestAddr :: binary(),
159     SrcPAN :: binary(),
160     SrcAddr :: binary(),
161     Payload :: binary().
162 decode_mac_header(
163     ?EXTENDED,
164     ?EXTENDED,
165     ?DISABLED,
166     <<DestPAN:16/bitstring, DestAddr:64/bitstring, SrcPAN:16/bitstring, SrcAddr
167     :64/bitstring, Payload/bitstring>>
168 ) ->
169     {DestPAN, DestAddr, SrcPAN, SrcAddr, Payload};
170 decode_mac_header(
171     ?EXTENDED,
172     ?EXTENDED,
173     ?ENABLED,
174     <<DestPAN:16/bitstring, DestAddr:64/bitstring, SrcAddr:64/bitstring, Payload/
175     bitstring>>
176 ) ->
177     {DestPAN, DestAddr, DestPAN, SrcAddr, Payload};
178 decode_mac_header(
179     ?EXTENDED,
180     ?SHORT_ADDR,
181     ?DISABLED,
182     <<DestPAN:16/bitstring, DestAddr:64/bitstring, SrcPAN:16/bitstring, SrcAddr
183     :16/bitstring, Payload/bitstring>>
184 ) ->

```

```

181     {DestPAN, DestAddr, SrcPAN, SrcAddr, Payload};
182 decode_mac_header(
183     ?EXTENDED,
184     ?SHORT_ADDR,
185     ?ENABLED,
186     <<DestPAN:16/bitstring, DestAddr:64/bitstring, SrcAddr:16/bitstring, Payload/
        bitstring>>
187 ) ->
188     {DestPAN, DestAddr, DestPAN, SrcAddr, Payload};
189 decode_mac_header(
190     ?EXTENDED,
191     ?NONE,
192     -,
193     <<DestPAN:16/bitstring, DestAddr:64/bitstring, Payload/bitstring>>
194 ) ->
195     {DestPAN, DestAddr, <<>>, <<>>, Payload};
196 decode_mac_header(
197     ?SHORT_ADDR,
198     ?EXTENDED,
199     ?DISABLED,
200     <<DestPAN:16/bitstring, DestAddr:16/bitstring, SrcPAN:16/bitstring, SrcAddr
        :64/bitstring, Payload/bitstring>>
201 ) ->
202     {DestPAN, DestAddr, SrcPAN, SrcAddr, Payload};
203 decode_mac_header(
204     ?SHORT_ADDR,
205     ?EXTENDED,
206     ?ENABLED,
207     <<DestPAN:16/bitstring, DestAddr:16/bitstring, SrcAddr:64/bitstring, Payload/
        bitstring>>
208 ) ->
209     {DestPAN, DestAddr, DestPAN, SrcAddr, Payload};
210 decode_mac_header(
211     ?SHORT_ADDR,
212     ?SHORT_ADDR,
213     ?DISABLED,
214     <<DestPAN:16/bitstring, DestAddr:16/bitstring, SrcPAN:16/bitstring, SrcAddr
        :16/bitstring, Payload/bitstring>>
215 ) ->
216     {DestPAN, DestAddr, SrcPAN, SrcAddr, Payload};
217 decode_mac_header(
218     ?SHORT_ADDR,
219     ?SHORT_ADDR,
220     ?ENABLED,
221     <<DestPAN:16/bitstring, DestAddr:16/bitstring, SrcAddr:16/bitstring, Payload/
        bitstring>>
222 ) ->
223     {DestPAN, DestAddr, DestPAN, SrcAddr, Payload};
224 decode_mac_header(
225     ?SHORT_ADDR,
226     ?NONE,
227     -,
228     <<DestPAN:16/bitstring, DestAddr:16/bitstring, Payload/bitstring>>
229 ) ->
230     {DestPAN, DestAddr, <<>>, <<>>, Payload};
231 decode_mac_header(
232     ?NONE,
233     ?EXTENDED,
234     -,
235     <<SrcPAN:16/bitstring, SrcAddr:64/bitstring, Payload/bitstring>>
236 ) ->
237     {<<>>, <<>>, SrcPAN, SrcAddr, Payload};

```

```

238 decode_mac_header(
239     ?NONE,
240     ?SHORT_ADDR,
241     -,
242     <<SrcPAN:16/bitstring, SrcAddr:16/bitstring, Payload/bitstring>>
243 ) ->
244     {<<>>, <<>>, SrcPAN, SrcAddr, Payload};
245 decode_mac_header(_SrcAddrMode, _DestAddrMode, _PanIdCompr, _Bits) ->
246     error(internal_decoding_error).
247
248 %-----
249 % @private
250 % @doc Creates a MAC frame control
251 % @param FrameType: MAC frame type
252 % @param AR: ACK request
253 % @end
254 %-----
255 -spec build_frame_control(FrameControl) -> <<_:16>> when FrameControl ::
    frame_control().
256 build_frame_control(FrameControl) ->
257     #frame_control{
258         pan_id_compr = PanIdCompr,
259         ack_req = AckReq,
260         frame_pending = FramePending,
261         sec_en = SecEn,
262         frame_type = FrameType,
263         src_addr_mode = SrcAddrMode,
264         frame_version = FrameVersion,
265         dest_addr_mode = DestAddrMode
266     } =
267         FrameControl,
268         <<2#0:1, PanIdCompr:1, AckReq:1, FramePending:1, SecEn:1, FrameType:3,
            SrcAddrMode:2, FrameVersion:2, DestAddrMode:2, 2#0:2>>.
269
270 %-----
271 % @private
272 % @doc Decode the frame control given in a bitstring form in the parameters
273 % @end
274 %-----
275 -spec decode_frame_control(FC) -> frame_control() when FC :: <<_:16>>.
276 decode_frame_control(FC) ->
277     <<_:1, PanIdCompr:1, AckReq:1, FramePending:1, SecEn:1, FrameType:3,
            SrcAddrMode:2, FrameVersion:2, DestAddrMode:2, _:2>> =
278         FC,
279         #frame_control{
280             frame_type = FrameType,
281             sec_en = SecEn,
282             frame_pending = FramePending,
283             ack_req = AckReq,
284             pan_id_compr = PanIdCompr,
285             dest_addr_mode = DestAddrMode,
286             frame_version = FrameVersion,
287             src_addr_mode = SrcAddrMode
288         }.
289
290 %--- Tool functions -----
291
292 % reverse_byte_order(Bitstring) ->
293 %     Size = bit_size(Bitstring),
294 %     <<X:Size/integer-little>> = Bitstring,
295 %     <<X:Size/integer-big>>.
296 reverse_byte_order(Bitstring) ->

```

```

297     reverse_byte_order(Bitstring, <<>>).
298
299 reverse_byte_order(<<>>, Acc) ->
300     Acc;
301 reverse_byte_order(<<Head:8>>, Acc) ->
302     <<Head:8, Acc/bitstring>>;
303 reverse_byte_order(<<Head:8, Tail/bitstring>>, Acc) ->
304     reverse_byte_order(Tail, <<Head:8, Acc/bitstring>>).

```

```

1 -module(mac_frame).
2
3 -include("mac_frame.hrl").
4
5 -export([encode/2]).
6 -export([encode/3]).
7 -export([encode_ack/2]).
8 -export([decode/1]).
9
10 %-----
11 % @doc builds a mac frame without a payload
12 % @equiv encode(FrameControl, MacHeader, <<>>)
13 % @end
14 %-----
15 -spec encode(FrameControl :: #frame_control{}, MacHeader :: #mac_header{}) ->
16     bitstring().
17 encode(FrameControl, MacHeader) ->
18     encode(FrameControl, MacHeader, <<>>).
19
20 %-----
21 % @doc builds a mac frame
22 % @returns a MAC frame ready to be transmitted in a bitstring (not including the
23 % CRC automatically added by the DW1000)
24 % @end
25 %-----
26 -spec encode(
27     FrameControl :: frame_control(),
28     MacHeader :: mac_header(),
29     Payload :: bitstring()
30 ) ->
31     bitstring().
32 encode(FrameControl, MacHeader, Payload) ->
33     Header = build_mac_header(FrameControl, MacHeader),
34     <<Header/bitstring, Payload/bitstring>>.
35
36 %-----
37 % @doc Builds an ACK frame
38 % @returns a MAC frame ready to be transmitted in a bitstring (not including the
39 % CRC automatically added by the DW1000)
40 % @end
41 %-----
42 encode_ack(FramePending, Seqnum) ->
43     FC = build_frame_control(#frame_control{
44         frame_type = ?FTYPE_ACK,
45         frame_pending = FramePending,
46         dest_addr_mode = ?NONE,
47         src_addr_mode = ?NONE
48     }),
49     <<FC/bitstring, Seqnum:8>>.
46
47 %-----
48 % @doc builds a mac header based on the FrameControl and the MacHeader structures

```

```

50     given in the args.
51 % <b> The MAC header doesn't support security fields yet </b>
52 % @returns the MAC header in a bitstring
53 % @end
54 -----
55 -spec build_mac_header(FrameControl, MacHeader) -> binary() when
56     FrameControl :: frame_control(),
57     MacHeader :: mac_header().
58 build_mac_header(FrameControl, MacHeader) ->
59     FC = build_frame_control(FrameControl),
60     DestPan = reverse_byte_order(MacHeader#mac_header.dest_pan),
61     DestAddr = reverse_byte_order(MacHeader#mac_header.dest_addr),
62     DestAddrFields =
63         case FrameControl#frame_control.dest_addr_mode of
64             ?NONE ->
65                 <<>>;
66             - ->
67                 <<DestPan/bitstring, DestAddr/bitstring>>
68         end,
69     SrcPan = reverse_byte_order(MacHeader#mac_header.src_pan),
70     SrcAddr = reverse_byte_order(MacHeader#mac_header.src_addr),
71     SrcAddrFields =
72         case
73             {
74                 FrameControl#frame_control.src_addr_mode,
75                 FrameControl#frame_control.pan_id_compr,
76                 FrameControl#frame_control.dest_addr_mode
77             }
78         of
79             {?NONE, _, _} ->
80                 <<>>;
81             {_, ?DISABLED, _} ->
82                 <<SrcPan/bitstring,
83                     % if no compression is applied on PANID and SRC addr is
84                     present
85                     SrcAddr/bitstring>>;
86             {_, ?ENABLED, ?NONE} ->
87                 <<SrcPan/bitstring,
88                     % if there is a compression of the PANID but the dest addr isn
89                     't present
90                     SrcAddr/bitstring>>;
91             {_, ?ENABLED, _} ->
92                 % if there is a compression of the PANID and the dest addr is
93                 present
94                 <<SrcAddr/bitstring>>
95         end,
96     <<FC/bitstring, (MacHeader#mac_header.seqnum):8, DestAddrFields/bitstring,
97     SrcAddrFields/bitstring>>.
98 -----
99 % @doc decodes the MAC frame given in the arguments
100 % @return A tuple containing the decoded frame control, the decoded mac header and
101 % the payload
102 % @end
103 -----
104 -spec decode(Data) -> {FrameControl, MacHeader, Payload} when
105     Data :: binary(),
106     FrameControl :: frame_control(),
107     MacHeader :: mac_header(),
108     Payload :: bitstring().

```

```

106 decode(Data) ->
107   <<FC:16/bitstring, Seqnum:8, Rest/bitstring>> = Data,
108   FrameControl = decode_frame_control(FC),
109   decode_rest(FrameControl, Seqnum, Rest).
110
111 %-----
112 % @private
113 % @doc Decodes the remaining sequence of bit present in the payload after the
114 % seqnum
115 % @end
116 %-----
117 -spec decode_rest(
118   FrameControl :: frame_control(),
119   Seqnum :: integer(),
120   Rest :: binary()
121 ) ->
122   {FrameControl :: frame_control(), MacHeader :: mac_header(), Payload :: binary
123     ()}.
124 decode_rest(
125   #frame_control{frame_type = ?FTYPE_ACK} = FrameControl,
126   Seqnum,
127   % Might cause an issue if piggybacking is used (allowed in IEEE 802.15.4?)
128   _Rest
129 ) ->
130   {FrameControl, #mac_header{seqnum = Seqnum}, <<>>};
131 decode_rest(FrameControl, Seqnum, Rest) ->
132   {DestPAN, DestAddr, SrcPAN, SrcAddr, Payload} =
133     decode_mac_header(
134       FrameControl#frame_control.dest_addr_mode,
135       FrameControl#frame_control.src_addr_mode,
136       FrameControl#frame_control.pan_id_compr,
137       Rest
138     ),
139     MacHeader =
140       #mac_header{
141         seqnum = Seqnum,
142         dest_pan = reverse_byte_order(DestPAN),
143         dest_addr = reverse_byte_order(DestAddr),
144         src_pan = reverse_byte_order(SrcPAN),
145         src_addr = reverse_byte_order(SrcAddr)
146       },
147     {FrameControl, MacHeader, Payload}.
148 % Note extended addresses and PAN ID are used in the case of inter-PAN
149 % communication
150 % In inter PAN communication, it can be omitted but it's not mandatory
151 -spec decode_mac_header(DestAddrMode, SrcAddrMode, PanIdCompr, Bits) ->
152   {DestPAN, DestAddr, SrcPAN, SrcAddr, Payload}
153 when
154   DestAddrMode :: flag(),
155   SrcAddrMode :: flag(),
156   PanIdCompr :: flag(),
157   Bits :: bitstring(),
158   DestPAN :: binary(),
159   DestAddr :: binary(),
160   SrcPAN :: binary(),
161   SrcAddr :: binary(),
162   Payload :: binary().
163 decode_mac_header(
164   ?EXTENDED,
165   ?EXTENDED,
166   ?DISABLED,

```



```

165     <<DestPAN:16/bitstring, DestAddr:64/bitstring, SrcPAN:16/bitstring, SrcAddr
166 ) ->
167     {DestPAN, DestAddr, SrcPAN, SrcAddr, Payload};
168 decode_mac_header(
169     ?EXTENDED,
170     ?EXTENDED,
171     ?ENABLED,
172     <<DestPAN:16/bitstring, DestAddr:64/bitstring, SrcAddr:64/bitstring, Payload/
173 ) ->
174     {DestPAN, DestAddr, DestPAN, SrcAddr, Payload};
175 decode_mac_header(
176     ?EXTENDED,
177     ?SHORT_ADDR,
178     ?DISABLED,
179     <<DestPAN:16/bitstring, DestAddr:64/bitstring, SrcPAN:16/bitstring, SrcAddr
180 ) ->
181     {DestPAN, DestAddr, SrcPAN, SrcAddr, Payload};
182 decode_mac_header(
183     ?EXTENDED,
184     ?SHORT_ADDR,
185     ?ENABLED,
186     <<DestPAN:16/bitstring, DestAddr:64/bitstring, SrcAddr:16/bitstring, Payload/
187 ) ->
188     {DestPAN, DestAddr, DestPAN, SrcAddr, Payload};
189 decode_mac_header(
190     ?EXTENDED,
191     ?NONE,
192     -,
193     <<DestPAN:16/bitstring, DestAddr:64/bitstring, Payload/bitstring>>
194 ) ->
195     {DestPAN, DestAddr, <<>>, <<>>, Payload};
196 decode_mac_header(
197     ?SHORT_ADDR,
198     ?EXTENDED,
199     ?DISABLED,
200     <<DestPAN:16/bitstring, DestAddr:16/bitstring, SrcPAN:16/bitstring, SrcAddr
201 ) ->
202     {DestPAN, DestAddr, SrcPAN, SrcAddr, Payload};
203 decode_mac_header(
204     ?SHORT_ADDR,
205     ?EXTENDED,
206     ?ENABLED,
207     <<DestPAN:16/bitstring, DestAddr:16/bitstring, SrcAddr:64/bitstring, Payload/
208 ) ->
209     {DestPAN, DestAddr, DestPAN, SrcAddr, Payload};
210 decode_mac_header(
211     ?SHORT_ADDR,
212     ?SHORT_ADDR,
213     ?DISABLED,
214     <<DestPAN:16/bitstring, DestAddr:16/bitstring, SrcPAN:16/bitstring, SrcAddr
215 ) ->
216     {DestPAN, DestAddr, SrcPAN, SrcAddr, Payload};
217 decode_mac_header(
218     ?SHORT_ADDR,
219     ?SHORT_ADDR,

```

```

220     ?ENABLED,
221     <<DestPAN:16/bitstring, DestAddr:16/bitstring, SrcAddr:16/bitstring, Payload/
        bitstring>>
222 ) ->
223     {DestPAN, DestAddr, DestPAN, SrcAddr, Payload};
224 decode_mac_header(
225     ?SHORT_ADDR,
226     ?NONE,
227     -,
228     <<DestPAN:16/bitstring, DestAddr:16/bitstring, Payload/bitstring>>
229 ) ->
230     {DestPAN, DestAddr, <<>>, <<>>, Payload};
231 decode_mac_header(
232     ?NONE,
233     ?EXTENDED,
234     -,
235     <<SrcPAN:16/bitstring, SrcAddr:64/bitstring, Payload/bitstring>>
236 ) ->
237     {<<>>, <<>>, SrcPAN, SrcAddr, Payload};
238 decode_mac_header(
239     ?NONE,
240     ?SHORT_ADDR,
241     -,
242     <<SrcPAN:16/bitstring, SrcAddr:16/bitstring, Payload/bitstring>>
243 ) ->
244     {<<>>, <<>>, SrcPAN, SrcAddr, Payload};
245 decode_mac_header(_SrcAddrMode, _DestAddrMode, _PanIdCompr, _Bits) ->
246     error(internal_decoding_error).
247
248 %-----
249 % @private
250 % @doc Creates a MAC frame control
251 % @param FrameType: MAC frame type
252 % @param AR: ACK request
253 % @end
254 %-----
255 -spec build_frame_control(FrameControl) -> <<_:16>> when FrameControl ::
        frame_control().
256 build_frame_control(FrameControl) ->
257     #frame_control{
258         pan_id_compr = PanIdCompr,
259         ack_req = AckReq,
260         frame_pending = FramePending,
261         sec_en = SecEn,
262         frame_type = FrameType,
263         src_addr_mode = SrcAddrMode,
264         frame_version = FrameVersion,
265         dest_addr_mode = DestAddrMode
266     } =
267         FrameControl,
268         <<2#0:1, PanIdCompr:1, AckReq:1, FramePending:1, SecEn:1, FrameType:3,
            SrcAddrMode:2, FrameVersion:2, DestAddrMode:2, 2#0:2>>.
269
270 %-----
271 % @private
272 % @doc Decode the frame control given in a bitstring form in the parameters
273 % @end
274 %-----
275 -spec decode_frame_control(FC) -> frame_control() when FC :: <<_:16>>.
276 decode_frame_control(FC) ->
277     <<_:1, PanIdCompr:1, AckReq:1, FramePending:1, SecEn:1, FrameType:3,
        SrcAddrMode:2, FrameVersion:2, DestAddrMode:2, _:2>> =

```

```

278     FC,
279     #frame_control{
280         frame_type = FrameType,
281         sec_en = SecEn,
282         frame_pending = FramePending,
283         ack_req = AckReq,
284         pan_id_compr = PanIdCompr,
285         dest_addr_mode = DestAddrMode,
286         frame_version = FrameVersion,
287         src_addr_mode = SrcAddrMode
288     }.
289
290 %--- Tool functions -----
291
292 % reverse_byte_order(Bitstring) ->
293 %     Size = bit_size(Bitstring),
294 %     <<X:Size/integer-little>> = Bitstring,
295 %     <<X:Size/integer-big>>.
296 reverse_byte_order(Bitstring) ->
297     reverse_byte_order(Bitstring, <<>>).
298
299 reverse_byte_order(<<>>, Acc) ->
300     Acc;
301 reverse_byte_order(<<Head:8>>, Acc) ->
302     <<Head:8, Acc/bitstring>>;
303 reverse_byte_order(<<Head:8, Tail/bitstring>>, Acc) ->
304     reverse_byte_order(Tail, <<Head:8, Acc/bitstring>>).

```

```

1 -module(pmod_uwb).
2 -behaviour(gen_server).
3
4 % API
5 -export([start_link/2]).
6 -export([read/1, write/2, write_tx_data/1, get_received_data/0, transmit/1,
7         transmit/2, wait_for_transmission/0, reception/0, reception/1]).
8 -export([reception_async/0]).
9 -export([set_frame_timeout/1]).
10 -export([set_preamble_timeout/1, disable_preamble_timeout/0]).
11 -export([softreset/0, clear_rx_flags/0]).
12 -export([disable_rx/0]).
13 -export([suspend_frame_filtering/0, resume_frame_filtering/0]).
14 -export([signal_power/0]).
15 -export([prf_value/0]).
16 -export([rx_preamble_repetition/0]).
17 -export([rx_data_rate/0]).
18 -export([rx_ranging_info/0]).
19 -export([std_noise/0]).
20 -export([first_path_power_level/0]).
21 -export([get_conf/0]).
22 -export([get_rx_metadata/0]).
23
24 % gen_server callback
25 -export([init/1, handle_call/3, handle_cast/2]).
26
27 -compile({nowarn_unused_function, [debug_read/2, debug_write/2, debug_write/3,
28         debug_bitstring/1, debug_bitstring_hex/1]}).
29
30 % Includes
31 -include("grisp.hrl").
32
33 -include("pmod_uwb.hrl").

```

```

32
33 %--- Macros -----
34
35 % Define the polarity and the phase of the clock
36 -define(SPI_MODE, #{clock => {low, leading}}).
37
38 -define(WRITE_ONLY_REG_FILE(RegFileID), RegFileID == tx_buffer).
39
40 -define(READ_ONLY_REG_FILE(RegFileID), RegFileID==dev_id;
41         RegFileID==sys_time;
42         RegFileID==rx_finfo;
43         RegFileID==rx_buffer;
44         RegFileID==rx_fqual;
45         RegFileID==rx_ttcko;
46         RegFileID==rx_time;
47         RegFileID==tx_time;
48         RegFileID==sys_state;
49         RegFileID==acc_mem).
50
51 %% The configurations of the subregisters of these register files are different
52 %% (some sub-registers are RO, some are RW and some have reserved bytes
53 %% that can't be written)
54 %% Thus, some registers files require to write their sub-register independently
55 %% => Write the sub-registers one by one instead of writing
56 %% the whole register file directly
57 -define(IS_SRW(RegFileID), RegFileID==agc_ctrl;
58         RegFileID==ext_sync;
59         RegFileID==ec_ctrl;
60         RegFileID==gpio_ctrl;
61         RegFileID==drx_conf;
62         RegFileID==rf_conf;
63         RegFileID==tx_cal;
64         RegFileID==fs_ctrl;
65         RegFileID==aon;
66         RegFileID==otp_if;
67         RegFileID==lde_if;
68         RegFileID==dig_diag;
69         RegFileID==pmsc).
70
71 -define(READ_ONLY_SUB_REG(SubRegister), SubRegister==irqs;
72         SubRegister==agc_stat1;
73         SubRegister==ec_rxtc;
74         SubRegister==ec_glop;
75         SubRegister==drx_car_int;
76         SubRegister==rf_status;
77         SubRegister==tc_sarl;
78         SubRegister==sarw;
79         SubRegister==tc_pg_status;
80         SubRegister==lde_thresh;
81         SubRegister==lde_ppindx;
82         SubRegister==lde_ppampl;
83         SubRegister==evc_phe;
84         SubRegister==evc_rse;
85         SubRegister==evc_fcg;
86         SubRegister==evc_fce;
87         SubRegister==evc_ffr;
88         SubRegister==evc_ovr;
89         SubRegister==evc_sto;
90         SubRegister==evc_pto;
91         SubRegister==evc_fwto;
92         SubRegister==evc_txfs;
93         SubRegister==evc_hpw;

```

```

94         SubRegister==evc_tpw).
95
96
97 %--- Types -----
98 -export_type([register_values/0]).
99
100 -type regFileID() :: atom().
101 -opaque register_values() :: map().
102
103 %--- API -----
104
105 start_link(Connector, _Opts) ->
106     gen_server:start_link({local, ?MODULE}, ?MODULE, Connector, []).
107
108
109 %% @doc read a register file
110 %%
111 %% === Example ===
112 %% To read the register file DEV_ID
113 %% '''
114 %% 1> pmod_uwb:read(dev_id).
115 %% #{model => 1, rev => 0, ridtag => "DECA", ver => 3}
116 %% '''
117 -spec read(RegFileID) -> Result when
118     RegFileID :: regFileID(),
119     Result    :: map() | {error, any()}.
120 read(RegFileID) when ?WRITE_ONLY_REG_FILE(RegFileID) ->
121     error({read_on_write_only_register, RegFileID});
122 read(RegFileID) -> call({read, RegFileID}).
123
124 %% @doc Write values in a register
125 %%
126 %% === Examples ===
127 %% To write in a simple register file (i.e. a register without any sub-register)
128 %% '''
129 %% 1> pmod_uwb:write(eui, #{eui => <<16#AAAAAABBBBBBBBBB>>}).
130 %% ok
131 %% '''
132 %% To write in one sub-register of a register file:
133 %% '''
134 %% 2> pmod_uwb:write(panadr, #{pan_id => <<16#AAAA>>}).
135 %% ok
136 %% '''
137 %% The previous code will only change the values inside the sub-register PAN_ID
138 %%
139 %% To write in multiple sub-register of a register file in the same burst:
140 %% '''
141 %% 3> pmod_uwb:write(panadr, #{pan_id => <<16#AAAA>>,
142 %%                          short_addr => <<16#BBBB>>}).
143 %% ok
144 %% '''
145 %% Some sub-registers have their own fields. For example to set the value of
146 %% the DIS_AM field in the sub-register AGC_CTRL1 of the register file AGC_CTRL:
147 %% '''
148 %% 4> pmod_uwb:write(agc_ctrl, #{agc_ctrl1 => #{dis_am => 2#0}}).
149 %% '''
150 -spec write(RegFileID, Value) -> Result when
151     RegFileID :: regFileID(),
152     Value     :: map(),
153     Result    :: ok | {error, any()}.
154 write(RegFileID, Value) when ?READ_ONLY_REG_FILE(RegFileID) ->
155     error({write_on_read_only_register, RegFileID, Value});

```

```

156 write(RegFileID, Value) when is_map(Value) ->
157     call({write, RegFileID, Value}).
158
159 %% @doc Writes the data in the TX_BUFFER register
160 %%
161 %% Value is expected to be a <b>Binary</b>
162 %% That choice was made to make the transmission of frames easier later on
163 %%
164 %% === Examples ===
165 %% Send "Hello" in the buffer
166 %% '''
167 %% 1> pmod_uwb:write_tx_data(<<"Hello">>).
168 %% '''
169 -spec write_tx_data(Value) -> Result when
170     Value  :: binary(),
171     Result :: ok | {error, any()}.
172 write_tx_data(Value) -> call({write_tx, Value}).
173
174 %% @doc Retrieves the data received on the UWB antenna
175 %% @returns {DataLength, Data}
176 -spec get_received_data() -> Result when
177     Result :: {integer(), bitstring()} | {error, any()}.
178 get_received_data() -> call({get_rx_data}).
179
180 get_rx_metadata() ->
181     #{rng := Rng} = read(rx_finfo),
182     #{rx_stamp := RxStamp} = read(rx_time),
183     #{tx_stamp := TxStamp} = read(tx_time),
184     #{rxtofs := Rxtofs} = read(rx_ttcko),
185     #{rxttcki := Rxttcki} = read(rx_tttcki),
186     #{snr => snr(),
187     prf => prf_value(),
188     pre => rx_preamble_repetition(),
189     data_rate => rx_data_rate(),
190     rng => Rng,
191     rx_stamp => RxStamp,
192     tx_stamp => TxStamp,
193     rxtofs => Rxtofs,
194     rxttcki => Rxttcki}.
195
196 % Source: https://forum.qorvo.com/t/how-to-calculate-the-signal-to-noise-ratio-snr-of-dw1000/5585/3
197 snr() ->
198     Delta = 87-7.5,
199     RSL = pmod_uwb:signal_power(),
200     RSL + Delta.
201
202 %% @doc Transmit data with the default options (i.e. don't wait for resp, ...)
203 %%
204 %% === Examples ===
205 %% To transmit a frame:
206 %% '''
207 %% 1> pmod_uwb:transmit(<<Version:4, NextHop:8>>).
208 %% ok.
209 %% '''
210 -spec transmit(Data) -> Result when
211     Data  :: bitstring(),
212     Result :: ok.
213 transmit(Data) when is_bitstring(Data) ->
214     call({transmit, Data, #tx_opts{}}),
215     wait_for_transmission().
216

```

```

217 %% @doc Performs a transmission with the specified options
218 %%
219 %% === Options ===
220 %% * wait4resp: It specifies that the reception must be enabled after
221 %%             the transmission in the expectation of a response
222 %% * w4r-tim: Specifies the turn around time in microseconds. That is the time
223 %%           the pmod will wait before enabling rx after a tx.
224 %%           Note that it won't be set if wit4resp is disabled
225 %% * txdllys: Specifies if the transmitter delayed sending should be set
226 %% * tx_delay: Specifies the delay of the transmission (see register DX_TIME)
227 %%
228 %% === Examples ===
229 %% To transmit a frame with default options:
230 %% '''
231 %% 1> pmod_uwb:transmit(<Version:4, NextHop:8>>, #tx_opts{}).
232 %% ok.
233 %% '''
234 -spec transmit(Data, Options) -> Result when
235     Data :: bitstring(),
236     Options :: tx_opts(),
237     Result :: ok.
238 transmit(Data, Options) ->
239     case Options#tx_opts.wait4resp of
240     ?ENABLED -> clear_rx_flags();
241     _ -> ok
242     end,
243     call({transmit, Data, Options}),
244     case read(sys_status) of
245     #{hdpwarn := 2#1} -> error({hdpwarn});
246     _ -> ok
247     end,
248     wait_for_transmission().
249
250 %% Wait for the transmission to be performed
251 %% usefull in the case of a delayed transmission
252 wait_for_transmission() ->
253     case read(sys_status) of
254     #{txfrs := 1} -> ok;
255     _ -> wait_for_transmission()
256     end.
257
258 %% @doc Receive data using the pmod
259 %% @equiv reception(false)
260 -spec reception() -> Result when
261     Result :: {integer(), bitstring()} | {error, any()}.
262 reception() ->
263     reception(false).
264
265 %% @doc Receive data using the pmod
266 %%
267 %% The function will hang until a frame is received on the board
268 %%
269 %% The CRC of the received frame <b>isn't</b> included in the returned value
270 %%
271 %% @param RXEnabled: specifies if the reception is already enabled on the board
272 %%                   (or set with delay)
273 %%
274 %% === Example ===
275 %% '''
276 %% 1> pmod_uwb:reception().
277 %% % Some frame is transmitted
278 %% {11, <<"Hello world">>}.

```

```

279 %% '''
280 -spec reception(RXEnabled) -> Result when
281     RXEnabled :: boolean(),
282     Result    :: {integer(), bitstring()} | {error, any()}.
283 reception(RXEnabled) ->
284     if not RXEnabled -> enable_rx();
285     true -> ok
286 end,
287 case wait_for_reception() of
288     ok ->
289         get_received_data();
290     Err ->
291         {error, Err}
292 end.
293
294 -spec reception_async() -> Result when
295     Result    :: ok | {error, any()}.
296 reception_async() ->
297     case reception() of
298         {error, _} = Err -> Err;
299     Frame ->
300         Metadata = get_rx_metadata(),
301         ieee802154_events:rx_event(Frame, Metadata)
302     end.
303
304 %% @private
305 enable_rx() ->
306     % io:format("Enabling reception~n"),
307     clear_rx_flags(),
308     call({write, sys_ctrl, #{rxenab => 2#1}}).
309
310 %% @doc Disables the reception on the pmod
311 disable_rx() ->
312     call({write, sys_ctrl, #{trxoff => 2#1}}).
313
314 wait_for_reception() ->
315     % io:format("Wait for resp~n"),
316     case read(sys_status) of
317         #{rxrfto := 1} -> rxrfto;
318         #{rxphe := 1} -> rxphe;
319         #{rxfce := 1} -> rxfce;
320         #{rxrfs1 := 1} -> rxrfs1;
321         #{rxpto := 1} -> rxpto;
322         #{rxsfdto := 1} -> rxsfdto;
323         #{ldeerr := 1} -> ldeerr;
324         #{affrej := 1} -> affrej;
325         #{rxdfc := 0} -> wait_for_reception();
326         #{rxfce := 1} -> rxfce;
327         #{rxfcg := 1} -> ok;
328         #{rxfcg := 0} -> wait_for_reception();
329         % #{rxdfc := 1, rxfcg := 1} -> ok; % The example driver doesn't do that
330         % but the user manual says that how you should check the reception of a
331         % frame
332         _ -> error({error_wait_for_reception})
333     end.
334
335 %% @doc Set the frame wait timeout and enables it
336 %% The unit is roughly 1us (cf. user manual)
337 %% If a float is given, it's decimal part is removed using trunc/1
338 %% @end
339 -spec set_frame_timeout(Timeout) -> Result when
340     Timeout :: microseconds(),

```



```

339     Result    :: ok.
340 set_frame_timeout(Timeout) when is_float(Timeout) ->
341     set_frame_timeout(trunc(Timeout));
342 set_frame_timeout(Timeout) when is_integer(Timeout) ->
343     write(rx_fwto, #{rxfwto => Timeout}),
344     write(sys_cfg, #{rxwtoc => 2#1}). % enable receive wait timeout
345
346 %% @doc Sets the preamble timeout. (PRETOC register of the DW1000)
347 %% The unit of 'Timeout' is in units usec
348 %% If the value is a float, trunc is called to remove the decimal part
349 %% Internally, it's converted in units of PAC size
350 -spec set_preamble_timeout(Timeout) -> ok when
351     Timeout :: non_neg_integer().
352 set_preamble_timeout(TO) when is_float(TO) ->
353     set_preamble_timeout(trunc(TO));
354 set_preamble_timeout(TO) when is_integer(TO) ->
355     call({preamble_timeout, TO}),
356     write(drx_conf, #{drx_pretoc => 0}).
357
358 disable_preamble_timeout() ->
359     write(drx_conf, #{drx_pretoc => 0}).
360
361 %% @doc Performs a reset of the IC following the procedure (cf. sec. 7.2.50.1)
362 softreset() ->
363     write(pmsc, #{pmsc_ctrl0 => #{sysclks => 2#01}}),
364     write(pmsc, #{pmsc_ctrl0 => #{softreset => 16#0}}),
365     write(pmsc, #{pmsc_ctrl0 => #{softreset => 16#FFFF}}).
366
367
368 clear_rx_flags() ->
369     write(sys_status, #{rxsfdto => 2#1,
370         rxpto => 2#1,
371         rxrfto => 2#1,
372         rxrfs1 => 2#1,
373         rxfce => 2#1,
374         rxphe => 2#1,
375         rxprd => 2#1,
376         rxdsfdd => 2#1,
377         rxphd => 2#1,
378         rxdfdr => 2#1,
379         rxfcg => 2#1}).
380
381 suspend_frame_filtering() ->
382     write(sys_cfg, #{ffcn => 2#0}).
383
384 resume_frame_filtering() ->
385     write(sys_cfg, #{ffcn => 2#1}).
386
387 %% @doc Returns the estimated value of the signal power in dBm
388 %% cf. user manual section 4.7.2
389 signal_power() ->
390     C = channel_impulse_resp_pow(), % Channel impulse response power value (
391         CIR_PWR)
392     A = case prf_value() of
393         16 -> 113.77;
394         64 -> 121.74
395     end, % Constant. For PRF of 16 MHz = 113.77, for PRF of 64MHz = 121.74
396     N = preamble_acc(), % Preamble accumulation count value (RXPACC but might be
397         adjusted)
398     % io:format("C: ~w-n A:~w-n N:~w-n", [C, A, N]),
399     Res = 10 * math:log10((C * math:pow(2, 17))/math:pow(N, 2)) - A,
400     % io:format("Estimated signal power: ~p dBm~n", [Res]),

```

```

399     % io:format("Std noise: ~w-n", [pmod_uwb:read(rx_fqual)]),
400     Res.
401
402 preamble_acc() ->
403     #{rxpacc := RXPACC} = read(rx_finfo),
404     #{rxpacc_nosat := RXPACC_NOSAT} = read(drx_conf),
405     if
406         RXPACC == RXPACC_NOSAT -> RXPACC - 5;
407         true -> RXPACC
408     end.
409
410 channel_impulse_resp_pow() ->
411     #{cir_pwr := CIR_PWR} = read(rx_fqual),
412     CIR_PWR.
413
414 %% @doc Gives the value of the PRF in MHz
415 -spec prf_value() -> 16 | 64.
416 prf_value() ->
417     #{agc_tune1 := AGC_TUNE1} = read(agc_ctrl),
418     case AGC_TUNE1 of
419         16#8870 -> 16;
420         16#889B -> 64
421     end.
422
423 %% @doc returns the preamble symbols repetition
424 rx_preamble_repetition() ->
425     #{rxpsr := RXPSR} = read(rx_finfo),
426     case RXPSR of
427         0 -> 16;
428         1 -> 64;
429         2 -> 1024;
430         3 -> 4096
431     end.
432
433 %% @doc returns the data rate of the received frame in kbps
434 rx_data_rate() ->
435     #{rxbr := RXBR} = read(rx_finfo),
436     case RXBR of
437         0 -> 110;
438         1 -> 850;
439         2 -> 6800
440     end.
441
442 % @doc returns the value of the 'Ranging' bit of the received frame
443 rx_ranging_info() ->
444     #{rng := RNG} = read(rx_finfo),
445     RNG.
446
447 std_noise() ->
448     #{std_noise := STD_NOISE} = read(rx_fqual),
449     STD_NOISE.
450
451 first_path_power_level() ->
452     #{fp_ampl1 := F1} = read(rx_time),
453     #{fp_ampl2 := F2, pp_ampl3 := F3} = read(rx_fqual),
454     A = 113.77,
455     N = preamble_acc(),
456     10 * math:log10((math:pow(F1,2) + math:pow(F2, 2) + math:pow(F3, 2))/math:pow(
457         N, 2)) - A.
458
459 get_conf() ->
460     call({get_conf}).

```

```

460
461 %--- gen_server Callbacks -----
462
463 %% @private
464 init(Slot) ->
465     % Verify the slot used
466     case {grisp_hw:platform(), Slot} of
467         {grisp2, spi2} -> ok;
468         {P, S} -> error({incompatible_slot, P, S})
469     end,
470     grisp_devices:register(Slot, ?MODULE),
471     Bus = grisp_spi:open(Slot),
472     case verify_id(Bus) of
473         ok -> softreset(Bus);
474         Val -> error({dev_id_no_match, Val})
475     end,
476     ldeload(Bus),
477     % TODO Merge the next 4 cfg commands into one
478     write_default_values(Bus),
479     config(Bus),
480     setup_sfd(Bus),
481     Conf = #phy_cfg{},
482     {ok, #{bus => Bus, conf => Conf}}.
483
484 %% @private
485 handle_call({read, RegFileID}, _From, #{bus := Bus} = State) ->
486     {reply, read_reg(Bus, RegFileID), State};
487 handle_call({write, RegFileID, Value}, _From, #{bus := Bus} = State) ->
488     {reply, write_reg(Bus, RegFileID, Value), State};
489 handle_call({write_tx, Value}, _From, #{bus := Bus} = State) ->
490     {reply, write_tx_data(Bus, Value), State};
491 handle_call({transmit, Data, Options}, _From, #{bus := Bus} = State) ->
492     {reply, tx(Bus, Data, Options), State};
493 handle_call({delayed_transmit, Data, Delay}, _From, #{bus := Bus} = State) ->
494     {reply, delayed_tx(Bus, Data, Delay), State};
495 handle_call({get_rx_data}, _From, #{bus := Bus} = State) ->
496     {reply, get_rx_data(Bus), State};
497 handle_call({get_conf}, _From, #{conf := Conf} = State) ->
498     {reply, Conf, State};
499 handle_call({preamble_timeout, T0us}, _From, State) ->
500     #{bus := Bus, conf := Conf} = State,
501     PACSize = Conf#phy_cfg.pac_size,
502     case T0us of
503         0 ->
504             write_reg(Bus, drx_conf, #{drx_pretoc => 0});
505         _ ->
506             % Remove 1 because DW1000 counter auto. adds 1 (cf. 7.2.40.9 user
507             % manual)
508             To = math:ceil(T0us / PACSize)-1,
509             write_reg(Bus, drx_conf, #{drx_pretoc => round(To)})
510     end,
511     {reply, ok, State};
512 handle_call(Request, _From, _State) ->
513     error({unknown_call, Request}).
514 %% @private
515 handle_cast(Request, _State) -> error({unknown_cast, Request}).
516
517 %--- Internal -----
518
519 call(Call) ->
520     Dev = grisp_devices:default(?MODULE),

```

```

521     gen_server:call(Dev#device.pid, Call).
522
523
524 %% @doc Verify the dev_id register of the pmod
525 %% @returns ok if the value is correct, otherwise the value read
526 verify_id(Bus) ->
527     #{ridtag := RIDTAG, model := MODEL} = read_reg(Bus, dev_id),
528     case {RIDTAG, MODEL} of
529         {"DECA", 1} -> ok;
530         _ -> {RIDTAG, MODEL}
531     end.
532
533 %% @private
534 %% Performs a softreset on the pmod
535 -spec softreset(Bus::grisp_spi:ref()) -> ok.
536 softreset(Bus) ->
537     write_reg(Bus, pmisc, #{pmisc_ctrl0 => #{sysclks => 2#01}}),
538     write_reg(Bus, pmisc, #{pmisc_ctrl0 => #{softrest => 16#0}}),
539     write_reg(Bus, pmisc, #{pmisc_ctrl0 => #{softreset => 16#FFFF}}).
540
541 %% @private
542 %% Writes the default values described in section 2.5.5 of the user manual
543 -spec write_default_values(Bus::grisp_spi:ref()) -> ok.
544 write_default_values(Bus) ->
545     write_reg(Bus, lde_if, #{lde_cfg1 => #{ntm => 16#D}, lde_cfg2 => 16#1607}),
546     write_reg(Bus, agc_ctrl, #{agc_tune1 => 16#8870, agc_tune2 => 16#2502A907}),
547     write_reg(Bus, drx_conf, #{drx_tune2 => 16#311A002D}),
548     write_reg(Bus, tx_power, #{tx_power => 16#0E082848}),
549     write_reg(Bus, rf_conf, #{rf_txctrl => 16#001E3FE3}),
550     write_reg(Bus, tx_cal, #{tc_pgdelay => 16#B5}),
551     write_reg(Bus, fs_ctrl, #{fs_plltune => 16#BE}).
552
553 %% @private
554 config(Bus) ->
555     write_reg(Bus, ext_sync, #{ec_ctrl => #{pllldt => 2#1}}),
556     write_reg(Bus, pmisc, #{pmisc_ctrl1 => #{lдерune => 2#0}}),
557     % Now enable RX and TX leds
558     write_reg(Bus, gpio_ctrl, #{gpio_mode => #{msgp2 => 2#01, msgp3 => 2#01}}),
559     % Enable RXOK and SFD leds
560     write_reg(Bus, gpio_ctrl, #{gpio_mode => #{msgp0 => 2#01, msgp1 => 2#01}}),
561     write_reg(Bus, pmisc, #{pmisc_ctrl0 => #{gpdce => 2#1, khzclken => 2#1}}),
562     write_reg(Bus, pmisc, #{pmisc_ledc => #{blnken => 2#1}}),
563     write_reg(Bus, dig_diag, #{evc_ctrl => #{evc_en => 2#1}}), % enable counting
564     % write_reg(Bus, sys_cfg, #{rxwtoe => 2#1}),
565     write_reg(Bus, tx_fctrl, #{txpsr => 2#10}). % Setting preamble symbols to 1024
566
567 %% @private
568 %% Load the microcode from ROM to RAM
569 %% It follows the steps described in section 2.5.5.10 of the DW1000 user manual
570 ldload(Bus) ->
571     write_reg(Bus, pmisc, #{pmisc_ctrl0 => #{sysclks => 2#01}}),
572     write_reg(Bus, pmisc, #{pmisc_ctrl0 => #{otp => 2#1, res8 => 2#1}}), % Writes 0
573     % x0301 in pmisc_ctrl0
574     write_reg(Bus, otp_if, #{otp_ctrl => #{ldload => 2#1}}), % Writes 0x8000 in
575     % OTP_CTRL
576     timer:sleep(150), % User manual requires a wait of 150 s
577     write_reg(Bus, pmisc, #{pmisc_ctrl0 => #{sysclks => 2#0}}), % Writes 0x0200 in
578     % pmisc_ctrl0
579     write_reg(Bus, pmisc, #{pmisc_ctrl0 => #{res8 => 2#0}}).
580
581 %% @private

```

```

579 %% If no frame is transmitted before AUTOACK, then the SFD isn't properly set
580 %% (cf. section 5.3.1.2 SFD initialisation)
581 setup_sfd(Bus) ->
582     write_reg(Bus, sys_ctrl, #{txsstr => 2#1, trxoff => 2#1}).
583
584 %% @private
585 %% Transmit the data using UWB
586 %% @param Options is used to set options about the transmission like a
587 %% transmission delay, etc.
587 -spec tx(grisp_spi:ref(), Data :: binary(), Options :: #tx_opts{}) -> ok.
588 tx(Bus, Data, #tx_opts{wait4resp = Wait4resp, w4r_tim = W4rTim, txdlys = TxDlys,
589     tx_delay = TxDelay, ranging = Ranging}) ->
589     % Writing the data that will be sent (w/o CRC)
590     DataLength = byte_size(Data) + 2, % DW1000 automatically adds the 2 bytes CRC
591     write_tx_data(Bus, Data),
592     % Setting the options of the transmission
593     case Wait4resp of
594         ?ENABLED -> write_reg(Bus, ack_resp_t, #{w4r_tim => W4rTim});
595         _ -> ok
596     end,
597     case TxDlys of
598         ?ENABLED -> write_reg(Bus, dx_time, #{dx_time => TxDelay});
599         _ -> ok
600     end,
601     write_reg(Bus, tx_fctrl, #{txboffs => 2#0, tr => Ranging, tflen => DataLength}
602     ),
603     write_reg(Bus, sys_ctrl, #{txsstr => 2#1, wait4resp => Wait4resp, txdlys =>
604     TxDlys}). % start transmission and some options
605
606 %% @private
607 %% Transmit the data with a specified delay using UWB
608 delayed_tx(Bus, Data, Delay) ->
609     write_reg(Bus, dx_time, #{dx_time => Delay}),
610     DataLength = byte_size(Data) + 2, % DW1000 automatically adds the 2 bytes CRC
611     write_tx_data(Bus, Data),
612     write_reg(Bus, tx_fctrl, #{txboffs => 2#0, tflen => DataLength}),
613     write_reg(Bus, sys_ctrl, #{txsstr => 2#1, txdlys => 2#1}). % start
614     transmission
615
616 %% @private
617 %% Get the received data (without the CRC bytes) stored in the rx_buffer
618 get_rx_data(Bus) ->
619     #{rxflen := FrameLength} = read_reg(Bus, rx_finfo),
620     Frame = read_rx_data(Bus, FrameLength-2), % Remove the CRC bytes
621     {FrameLength, Frame}.
622
623 %% @private
624 %% @doc Reverse the byte order of the bitstring given in the argument
625 %% @param Bin a bitstring
626 reverse(Bin) -> reverse(Bin, <<>>).
627 reverse(<<Bin:8>>, Acc) ->
628     <<Bin, Acc/binary>>;
629 reverse(<<Bin:8, Rest/bitstring>>, Acc) ->
630     reverse(Rest, <<Bin, Acc/binary>>).
631
632 % Source: https://stackoverflow.com/a/43310493
633 % reverse(Binary) ->
634 %     Size = bit_size(Binary),
635 %     <<X:Size/integer-little>> = Binary,
636 %     <<X:Size/integer-big>>.
637
638 %% @private

```

```

636 %% @doc Creates the header of the SPI transaction between the GRISP and the pmod
637 %%
638 %% It creates a header of 1 bytes. The header is used in a transaction that will
        affect
639 %% the whole register file (read/write)
640 %%
641 %% @param Op an atom (either <i>read</i> or <i>write</i>)
642 %% @param RegFileID an atom representing the register file
643 %% @returns a formatted header of <b>1 byte</b> long as described in the user
        manual
644 header(Op, RegFileID) ->
645     <<(rw(Op)):1, 2#0:1, (regFile(RegFileID)):6>>.
646
647 %% @private
648 %% @doc Creates the header of the SPI transaction between the GRISP and the pmod
649 %%
650 %% It creates a header of 2 bytes. The header is used in a transaction that will
        affect
651 %% the whole sub-register (read/write)
652 %% Careful: The sub-register needs to be mapped in the hrl file
653 %%
654 %% @param Op an atom (either <i>read</i> or <i>write</i>)
655 %% @param RegFileID an atom representing the register file
656 %% @param SubRegister an atom representing the sub-register
657 %% @returns a formatted header of <b>2 byte</b> long as described in the user
        manual
658 header(Op, RegFileID, SubRegister) ->
659     case subReg(SubRegister) < 127 of
660         true -> header(Op, RegFileID, SubRegister, 2);
661         _ -> header(Op, RegFileID, SubRegister, 3)
662     end.
663
664 header(Op, RegFileID, SubRegister, 2) ->
665     << (rw(Op)):1, 2#1:1, (regFile(RegFileID)):6,
666         2#0:1, (subReg(SubRegister)):7 >>;
667 header(Op, RegFileID, SubRegister, 3) ->
668     <<_:1, HighOrder:8, LowOrder:7>> = <<(subReg(SubRegister)):16>>,
669     << (rw(Op)):1, 2#1:1, (regFile(RegFileID)):6,
670         2#1:1, LowOrder:7,
671         HighOrder:8>>.
672
673 %% @private
674 %% @doc Read the values stored in a register file
675 read_reg(Bus, lde_ctrl) -> read_reg(Bus, lde_if);
676 read_reg(Bus, lde_if) ->
677     lists:foldl(fun(Elem, Acc) ->
678         Res = read_sub_reg(Bus, lde_if, Elem),
679         maps:merge(Acc, Res)
680     end,
681     #[],
682     [lde_thresh, lde_cfg1, lde_ppindx, lde_ppampl, lde_rxantd,
        lde_cfg2, lde_repc]);
683 read_reg(Bus, RegFileID) ->
684     Header = header(read, RegFileID),
685     [Resp] = grisp_spi:transfer(Bus, [{?SPI_MODE, Header, 1, regSize(RegFileID)}])
        ,
686     % debug_read(RegFileID, Resp),
687     reg(decode, RegFileID, Resp).
688
689
690 read_sub_reg(Bus, RegFileID, SubRegister) ->
691     Header = header(read, RegFileID, SubRegister),

```

```

692     HeaderSize = byte_size(Header),
693     % io:format("[HEADER] type ~w - ~w - ~w~n", [HeaderSize, Header, subRegSize(
        SubRegister)]),
694     [Resp] = grisp_spi:transfer(Bus, [{?SPI_MODE, Header, HeaderSize, subRegSize(
        SubRegister)}]),
695     reg(decode, SubRegister, Resp).
696
697
698 %% @doc get the received data
699 %% @param Length is the total length of the data we are trying to read
700 read_rx_data(Bus, Length) ->
701     Header = header(read, rx_buffer),
702     [Resp] = grisp_spi:transfer(Bus, [{?SPI_MODE, Header, 1, Length}]),
703     Resp.
704
705 % TODO: check that user isn't trying to write reserved bits by passing res, res1,
        ... in the map fields
706 %% @doc used to write the values in the map given in the Value argument
707 -spec write_reg(Bus::grisp_spi:ref(), RegFileID::regFileID(), Value::map()) -> ok.
708 % Write each sub-register one by one.
709 % If the user tries to write in a read-only sub-register, an error is thrown
710 write_reg(Bus, RegFileID, Value) when ?IS_SRW(RegFileID) ->
711     maps:map(
712         fun(SubRegister, Val) ->
713             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
714             Body = case CurrVal of
715                 V when is_map(V) -> reg(encode, SubRegister, maps:
                    merge_with(fun(_Key, _Old, New) -> New end, CurrVal,
                    Val));
716                 _ -> reg(encode, SubRegister, #{SubRegister => Val})
717             end,
718             Header = header(write, RegFileID, SubRegister),
719             % debug_write(RegFileID, SubRegister, Body),
720             _ = grisp_spi:transfer(Bus, [{?SPI_MODE, <<Header/binary, Body/binary
                >>, 2+subRegSize(SubRegister), 0}]),
721             end,
722             Value),
723     ok;
724 write_reg(Bus, RegFileID, Value) ->
725     Header = header(write, RegFileID),
726     CurrVal = read_reg(Bus, RegFileID),
727     ValuesToWrite = maps:merge_with(fun(_Key, _Value1, Value2) -> Value2 end,
        CurrVal, Value),
728     Body = reg(encode, RegFileID, ValuesToWrite),
729     % debug_write(RegFileID, Body),
730     _ = grisp_spi:transfer(Bus, [{?SPI_MODE, <<Header/binary, Body/binary>>, 1+
        regSize(RegFileID), 0}]),
731     ok.
732
733 %% @doc write_tx_data/2 sends data (Value) in the register tx_buffer
734 %% @param Value is the data to be written. It must be a binary and have a size of
        maximum 1024 bits
735 write_tx_data(Bus, Value) when is_binary(Value), (bit_size(Value) < 1025) ->
736     Header = header(write, tx_buffer),
737     Length = byte_size(Value),
738     % debug_write(tx_buffer, Body),
739     _ = grisp_spi:transfer(Bus, [{?SPI_MODE, <<Header/binary, Value/binary>>, 1+
        Length, 0}]),
740     ok.
741

```

```

742 %---- Register mapping -----
743
744 %% @doc Used to either decode the data returned by the pmod or to encode to data
745 %% that will be sent to the pmod
746 %%
747 %% The transmission on the MISO line is done byte by byte starting from the lowest
748 %% rank byte to the highest rank
749 %% Example: dev_id value is 0xDECA0130 but 0x3001CADE is transmitted over the MISO
750 %% line
751 -spec reg(Type, Register, Val) -> Ret when
752   Type      :: encode | decode,
753   Register  :: regFileID(),
754   Val       :: nonempty_binary() | register_values(),
755   Ret       :: nonempty_binary() | register_values().
756 reg(encode, SubRegister, Value) when ?READ_ONLY_SUB_REG(SubRegister) -> error({
757   writing_read_only_sub_register, SubRegister, Value});
758 reg(decode, dev_id, Resp) ->
759   <<
760     RIDTAG:16, Model:8, Ver:4, Rev:4
761   >> = reverse(Resp),
762   #{
763     ridtag => integer_to_list(RIDTAG, 16), model => Model, ver => Ver, rev =>
764     Rev
765   };
766 reg(decode, eui, Resp) ->
767   #{
768     eui => reverse(Resp)
769   };
770 reg(encode, eui, Val) ->
771   #{
772     eui := EUI
773   } = Val,
774   reverse(
775     EUI
776   );
777 reg(decode, panadr, Resp) ->
778   <<
779     PanId:16, ShortAddr:16
780   >> = reverse(Resp),
781   #{
782     pan_id => <<PanId:16>>, short_addr => <<ShortAddr:16>>
783   };
784 reg(encode, panadr, Val) ->
785   #{
786     pan_id := PanId, short_addr := ShortAddr
787   } = Val,
788   reverse(<<
789     PanId:16/bitstring, ShortAddr:16/bitstring
790   >>);
791 reg(decode, sys_cfg, Resp) ->
792   <<
793     FFA4:1, FFAR:1, FFAM:1, FFAA:1, FFAD:1, FFAB:1, FFBC:1, FFEN:1, % bits 7-0
794     FCS_INIT2F:1, DIS_RSDE:1, DIS_PHE:1, DIS_DRXB:1, DIS_FCE:1, SPI_EDGE:1,
795     HIRQ_POL:1, FFA5:1, % bits 15-8
796     _:1, RXM110K:1, _:3, DIS_STXP:1, PHR_MODE:2, % bits 23-16
797     AACKPEND:1, AUTOACK:1, RXAUTR:1, RXWTOE:1, _:4 % bits 31-24
798   >> = Resp,
799   #{
800     aackpend => AACKPEND, autoack => AUTOACK, rxautr => RXAUTR, rxwtoe =>
801     RXWTOE,
802     rxm110k => RXM110K, dis_stxp => DIS_STXP, phr_mode => PHR_MODE,
803     fcs_init2f => FCS_INIT2F, dis_rsde => DIS_RSDE, dis_phe => DIS_PHE,

```



```

797         dis_drxb => DIS_DRXB, dis_fce => DIS_FCE, spi_edge => SPI_EDGE,
798         hirq_pol => HIRQ_POL, ffa5 => FFA5,
799     ffa4 => FFA4, ffar => FFAR, ffam => FFAM, ffaa => FFAA, ffad => FFAD, ffab
800     => FFAB, ffbc => FFBC, ffen => FFEN
801     };
802     reg(encode, sys_cfg, Val) ->
803     #{
804         aackpend := AACKPEND, autoack := AUTOACK, rxautr := RXAUTR, rxwtoe :=
805         RXWTOE,
806         rxm110k := RXM110K, dis_stxp := DIS_STXP, phr_mode := PHR_MODE,
807         fcs_init2f := FCS_INIT2F, dis_rsde := DIS_RSDE, dis_phe := DIS_PHE,
808         dis_drxb := DIS_DRXB, dis_fce := DIS_FCE, spi_edge := SPI_EDGE,
809         hirq_pol := HIRQ_POL, ffa5 := FFA5,
810         ffa4 := FFA4, ffar := FFAR, ffam := FFAM, ffaa := FFAA, ffad := FFAD, ffab
811         := FFAB, ffbc := FFBC, ffen := FFEN
812     } = Val,
813     <<
814     FFA4:1, FFAR:1, FFAM:1, FFAA:1, FFAD:1, FFAB:1, FFBC:1, FFEN:1, % bits 7-0
815     FCS_INIT2F:1, DIS_RSDE:1, DIS_PHE:1, DIS_DRXB:1, DIS_FCE:1, SPI_EDGE:1,
816     HIRQ_POL:1, FFA5:1, % bits 15-8
817     2#0:1, RXM110K:1, 2#0:3, DIS_STXP:1, PHR_MODE:2, % bits 23-16
818     AACKPEND:1, AUTOACK:1, RXAUTR:1, RXWTOE:1, 2#0:4 % bits 31-24
819     >>;
820     reg(decode, sys_time, Resp) ->
821     <<
822     SysTime:40
823     >> = reverse(Resp),
824     #{
825         sys_time => SysTime
826     };
827     reg(decode, tx_fctrl, Resp) ->
828     <<
829     IFSDELAY:8, TXBOFFS:10, PE:2, TXPSR:2, TXPRF:2, TR:1, TXBR:2, R:3, TFLE:3,
830     TFLEN:7
831     >> = reverse(Resp),
832     #{
833         ifsdelay => IFSDELAY, txboffs => TXBOFFS, pe => PE, txpsr => TXPSR, txprf
834         => TXPRF, tr => TR, txbr => TXBR, r => R, tfle => TFLE, tflen => TFLEN
835     };
836     reg(encode, tx_fctrl, Val) ->
837     #{
838         ifsdelay := IFSDELAY, txboffs := TXBOFFS, pe := PE, txpsr := TXPSR, txprf
839         := TXPRF, tr := TR, txbr := TXBR, r := R, tfle := TFLE, tflen := TFLEN
840     } = Val,
841     reverse(<<
842         IFSDELAY:8, TXBOFFS:10, PE:2, TXPSR:2, TXPRF:2, TR:1, TXBR:2, R:3, TFLE:3,
843         TFLEN:7
844     >>);
845     % TX_BUFFER is write only => no decode
846     reg(decode, dx_time, Resp) ->
847     #{
848         dx_time => reverse(Resp)
849     };
850     reg(encode, dx_time, Val) ->
851     #{
852         dx_time := DX_TIME
853     } = Val,
854     reverse(<<
855         DX_TIME:40
856     >>);
857     reg(decode, rx_fwto, Resp) ->
858     <<

```

```

847     RXFWTO:16
848     >> = reverse(Resp),
849     #{
850         rxfwto => RXFWTO
851     };
852 reg(encode, rx_fwto, Val) ->
853     #{
854         rxfwto := RXFWTO
855     } = Val,
856     reverse(<<
857         RXFWTO:16
858     >>);
859 reg(decode, sys_ctrl, Resp) ->
860     <<
861         WAIT4RESP:1, TRXOFF:1, _:2, CANSFCS:1, TXDLYS:1, TXSTRT:1, SFCST:1, % bits
862             7-0
863         _:6, RXDLYE:1, RXENAB:1, % bits 15-8
864         _:8, % bits 23-16
865         _:7, HRBPT:1 % bits 31-24
866     >> = Resp,
867     #{
868         sfcst => SFCST, txstrt => TXSTRT, txdlys => TXDLYS, cansfcs => CANSFCS,
869         trxoff => TRXOFF, wait4resp => WAIT4RESP,
870         rxenab => RXENAB, rxdlye => RXDLYE,
871         hrbpt => HRBPT
872     };
873 reg(encode, sys_ctrl, Val) ->
874     #{
875         sfcst := SFCST, txstrt := TXSTRT, txdlys := TXDLYS, cansfcs := CANSFCS,
876         trxoff := TRXOFF, wait4resp := WAIT4RESP,
877         rxenab := RXENAB, rxdlye := RXDLYE,
878         hrbpt := HRBPT
879     } = Val,
880     <<
881         WAIT4RESP:1, TRXOFF:1, 2#0:2, CANSFCS:1, TXDLYS:1, TXSTRT:1, SFCST:1, %
882             bits 7-0
883         2#0:6, RXDLYE:1, RXENAB:1, % bits 15-8
884         2#0:8, % bits 23-16
885         2#0:7, HRBPT:1 % bits 31-24
886     >>;
887 reg(decode, sys_mask, Resp) ->
888     <<
889         MTXFRS:1, MTXPHS:1, MTXPRS:1, MTXFRB:1, MAAT:1, MESYNCR:1, MCPLOCK:1,
890         Reserved0:1, % bits 7-0
891         MRXFCE:1, MRXFCG:1, MRXDFR:1, MRXPHE:1, MRXPHD:1, MLDEDON:1, MRXSFD:1,
892         MRXPRD:1, % bits 15-8
893         MSLP2INIT:1, MGPIOIRQ:1, MRXPTO:1, MRXOVRR:1, Reserved1:1, MLDEERR:1,
894         MRXRFTO:1, MRXRFSL:1, % bits 23-16
895         Reserved2:2, MAFFREJ:1, MTXBERR:1, MHPDDWAR:1, MPLLHILO:1, MCPLLLL:1,
896         MRFPLLLL:1 % bits 31-24
897     >> = Resp,
898     #{
899         mtxfrc => MTXFRS, mtxphe => MTXPHS, mtxphe => MTXPHS, mtxphe => MTXPHS,
900         maat => MAAT, mesyncr => MESYNCR, mcplck => MCPLOCK, res0 =>
901             Reserved0, % bits 7-0
902         mrxfce => MRXFCE, mrxfcg => MRXFCG, mrxdfr => MRXDFR, mrxphe => MRXPHE,
903         mrxphd => MRXPHD, mldeon => MLDEDON, mrxsfd => MRXSFD, mrxprd =>
904         MRXPRD, % bits 15-8
905         mslp2init => MSLP2INIT, mgpioirq => MGPIOIRQ, mrxpto => MRXPTO, mrxovrr =>
906         MRXOVRR, res1 => Reserved1, mldeerr => MLDEERR, mrxrfto => MRXRFTO,
907         mrxrfsl => MRXRFSL, % bits 23-16
908         res2 => Reserved2, maffrej => MAFFREJ, mtxberr => MTXBERR, mhpddwar =>

```

```

MHPDDWAR, mpllhilo => MPLLHILO, mcpllll => MCPLLLL, mrfpllll =>
MRFPLLLL % bits 31-24
895 };
896 reg(encode, sys_mask, Val) ->
897 #{
898     mtxfrs := MTXFRS, txpchs := TXPHS, txprs := TXPRS, txfrb := TXFRB,
      maat := MAAT, mesyncr := MESYNCR, mcplck := MCPLOCK, res0 :=
      Reserved0, % bits 7-0
899     mrxfce := MRXFCE, mrxfcg := MRXFCG, mrxdfc := MRXDFR, mrxphe := MRXPHE,
      mrxphd := MRXPHD, mldeon := MLDEDON, mrxsfdd := MRXSFD, mrxprd :=
      MRXPRD, % bits 15-8
900     mslp2init := MSLP2INIT, mgpioirq := MGPIORQ, mrxpto := MRXPTO, mrxovrr :=
      MRXOVRR, res1 := Reserved1, mldeerr := MLDEERR, mrxrfto := MRXRFTO,
      mrxrfs1 := MRXRFS1, % bits 23-16
901     res2 := Reserved2, maffrej := MAFFREJ, mxberr := MTXBERR, mhpddwar :=
      MHPDDWAR, mpllhilo := MPLLHILO, mcpllll := MCPLLLL, mrfpllll :=
      MRFPLLLL % bits 31-24
902 } = Val,
903 <<
904     MTXFRS:1, TXPHS:1, TXPRS:1, TXFRB:1, MAAT:1, MESYNCR:1, MCPLOCK:1,
      Reserved0:1, % bits 7-0
905     MRXFCE:1, MRXFCG:1, MRXDFR:1, MRXPHE:1, MRXPHD:1, MLDEDON:1, MRXSFD:1,
      MRXPRD:1, % bits 15-8
906     MSLP2INIT:1, MGPIORQ:1, MRXPTO:1, MRXOVRR:1, Reserved1:1, MLDEERR:1,
      MRXRFTO:1, MRXRFS1:1, % bits 23-16
907     Reserved2:2, MAFFREJ:1, MTXBERR:1, MHPDDWAR:1, MPLLHILO:1, MCPLLLL:1,
      MRFPLLLL:1 % bits 31-24
908 >>;
909 reg(decode, sys_status, Resp) ->
910 <<
911     TXFRS:1, TXPHS:1, TXPRS:1, TXFRB:1, AAT:1, ESYNCR:1, CPLOCK:1, IRQS:1, %
      bits 7-0
912     RXFCE:1, RXFCG:1, RXDFR:1, RXPHE:1, RXPHE:1, LDEDONE:1, RXSFD:1, RXPRD:1,
      % bits 15-8
913     SPL2INIT:1, GPIOIRQ:1, RXPTO:1, RXOVRR:1, Reserved0:1, LDEERR:1, RXRFTO:1,
      RXRFS1:1, % bits 23-16
914     ICRBP:1, HSRBP:1, AFFREJ:1, TXBERR:1, HPDWARN:1, RXSFDTO:1, CLCKPLL_LL:1,
      RFPLL_LL:1, % bits 31-24
915     Reserved1:5, TXPUTE:1, RXPREJ:1, RXRSCS:1 % bits 39-32
916 >> = Resp,
917 #{
918     txfrs => TXFRS, txpchs => TXPHS, txprs => TXPRS, txfrb => TXFRB, aat => AAT
      , esyncr => ESYNCR, cplock => CPLOCK, irqsc => IRQS, % bits 7-0
919     rxfce => RXFCE, rxfcg => RXFCG, rxdfr => RXDFR, rxphe => RXPHE, rxphd =>
      RXPHD, ldedone => LDEDONE, rxsfdd => RXSFD, rxprd => RXPRD, % bits
      15-8
920     spl2init => SPL2INIT, gpioirq => GPIOIRQ, rxpto => RXPTO, rxovrr =>
      RXOVRR, res0 => Reserved0, ldeerr => LDEERR, rxrfto => RXRFTO, rxrfs1
      => RXRFS1, % bits 23-16
921     icrbp => ICRBP, hsrbp => HSRBP, affrej => AFFREJ, txberr => TXBERR,
      hdpwarn => HPDWARN, rxsfdto => RXSFDTO, clkpll_ll => CLCKPLL_LL,
      rfpll_ll => RFPLL_LL, % bits 31-24
922     res1 => Reserved1, txpute => TXPUTE, rxprej => RXPREJ, rxrscs => RXRSCS
923 };
924 reg(encode, sys_status, Val) ->
925 #{
926     txfrs := TXFRS, txpchs := TXPHS, txprs := TXPRS, txfrb := TXFRB, aat := AAT
      , esyncr := ESYNCR, cplock := CPLOCK, irqsc := IRQS, % bits 7-0
927     rxfce := RXFCE, rxfcg := RXFCG, rxdfr := RXDFR, rxphe := RXPHE, rxphd :=
      RXPHD, ldedone := LDEDONE, rxsfdd := RXSFD, rxprd := RXPRD, % bits
      15-8
928     spl2init := SPL2INIT, gpioirq := GPIOIRQ, rxpto := RXPTO, rxovrr :=

```

```

RXOVR, res0 := Reserved0, ldeerr := LDEERR, rxrfto := RXRFTO, rxrfs1
:= RXRFSL, % bits 23-16
929 icrbp := ICRBP, hsrbp := HSRBP, affrej := AFFREJ, txberr := TXBERR,
hdwarn := HPDWARN, rxsfdto := RXSFDTO, clkpll_ll := CLCKPLL_LL,
rfpll_ll := RFPLL_LL, % bits 31-24
930 res1 := Reserved1, txpute := TXPUTE, rxprej := RXPREJ, rxrscs := RXRSCS
931 } = Val,
932 <<
933 TXFRS:1, TXPHS:1, TXPRS:1, TXFRB:1, AAT:1, ESYNCR:1, CPLOCK:1, IRQS:1, %
bits 7-0
934 RXFCE:1, RXFCG:1, RXDFR:1, RXPHE:1, RXPHE:1, RXPHE:1, LDEDONE:1, RXSFDD:1, RXPRD:1,
% bits 15-8
935 SPL2INIT:1, GPIOIRQ:1, RXPTO:1, RXOVR:1, Reserved0:1, LDEERR:1, RXRFTO:1,
RXRFSL:1, % bits 23-16
936 ICRBP:1, HSRBP:1, AFFREJ:1, TXBERR:1, HPDWARN:1, RXSFDTO:1, CLCKPLL_LL:1,
RFPLL_LL:1, % bits 31-24
937 Reserved1:5, TXPUTE:1, RXPREJ:1, RXRSCS:1 % bits 39-32
938 >>;
939 reg(decode, rx_finfo, Resp) ->
940 <<
941 RXPACC:12, RXPSR:2, RXPRFR:2, RNG:1, RXBR:2, RXNSPL:2, _:1, RXFLE:3,
RXFLEN:7
942 >> = reverse(Resp),
943 #{
944 rxpacc => RXPACC, rxpsr => RXPSR, rxprfr => RXPRFR, rng => RNG, rxbr =>
RXBR, rxnspl => RXNSPL, rxfle => RXFLE, rxflen => RXFLEN
945 };
946 reg(decode, rx_buffer, Resp) ->
947 #{ rx_buffer => reverse(Resp)};
948 reg(decode, rx_fqual, Resp) ->
949 <<
950 CIR_PWR:16, PP_AMPL3:16, FP_AMPL2:16, STD_NOISE:16
951 >> = Resp,
952 #{
953 cir_pwr => CIR_PWR, pp_ampl3 => PP_AMPL3, fp_ampl2 => FP_AMPL2, std_noise
=> STD_NOISE
954 };
955 reg(decode, rx_ttcki, Resp) ->
956 <<
957 RXTTCKI:32
958 >> = reverse(Resp),
959 #{
960 rxttcki => RXTTCKI
961 };
962 reg(decode, rx_ttcko, Resp) ->
963 <<
964 _:1, RCPHASE:7, RSMPDEL:8, _:5, RXTOFS:19
965 >> = reverse(Resp),
966 #{
967 rcphase => RCPHASE, rsmpdel => RSMPDEL, rxtofs => RXTOFS
968 };
969 reg(decode, rx_time, Resp) ->
970 <<
971 RX_RAWST:40, FP_AMPL1:16, FP_INDEX:16, RX_STAMP:40
972 >> = reverse(Resp),
973 #{
974 rx_rawst => RX_RAWST, fp_ampl1 => FP_AMPL1, fp_index => FP_INDEX, rx_stamp
=> RX_STAMP
975 };
976 reg(decode, tx_time, Resp) ->
977 <<
978 TX_RAWST:40, TX_STAMP:40

```

```

979     >> = reverse(Resp),
980     #{
981         tx_rawst => TX_RAWST, tx_stamp => TX_STAMP
982     };
983 reg(decode, tx_antd, Resp) ->
984     #{
985         tx_antd => reverse(Resp)
986     };
987 reg(encode, tx_antd, Val) ->
988     #{
989         tx_antd := TX_ANTD
990     } = Val,
991     reverse(<<
992         TX_ANTD:16
993     >>);
994 reg(decode, sys_state, Resp) ->
995     <<
996         _:8, _:4, PMSC_STATE:4, _:3, RX_STATE:5, _:4, TX_STATE:4
997     >> = reverse(Resp),
998     #{
999         pmsc_state => PMSC_STATE, rx_state => RX_STATE, tx_state => TX_STATE
1000 };
1001 reg(decode, ack_resp_t, Resp) ->
1002     <<
1003         ACK_TIME:8, _:4, W4R_TIME:20
1004     >> = reverse(Resp),
1005     #{
1006         ack_tim => ACK_TIME, w4r_tim => W4R_TIME
1007     };
1008 reg(encode, ack_resp_t, Val) ->
1009     #{
1010         ack_tim := ACK_TIME, w4r_tim := W4R_TIME
1011     } = Val,
1012     reverse(<<
1013         ACK_TIME:8, 2#0:4, W4R_TIME:20
1014     >>);
1015 reg(decode, rx_sniff, Resp) ->
1016     <<
1017         Reserved0:16, SNIFF_OFFT:8, Reserved1:4, SNIFF_ONT:4
1018     >> = reverse(Resp),
1019     #{
1020         res0 => Reserved0,
1021         sniff_offt => SNIFF_OFFT,
1022         sniff_ont => SNIFF_ONT,
1023         res1 => Reserved1
1024     };
1025 reg(encode, rx_sniff, Val) ->
1026     #{
1027         res0 := Reserved0,
1028         sniff_offt := SNIFF_OFFT,
1029         sniff_ont := SNIFF_ONT,
1030         res1 := Reserved1
1031     } = Val,
1032     reverse(<<
1033         Reserved0:16, SNIFF_OFFT:8, Reserved1:4, SNIFF_ONT:4
1034     >>);
1035 % Smart transmit power control (cf. user manual p 104)
1036 reg(decode, tx_power, Resp) ->
1037     <<
1038         BOOSTP125:8, BOOSTP250:8, BOOSTP500:8, BOOSTNORM:8
1039     >> = reverse(Resp),
1040     #{

```

```

1041     boostp125 => BOOSTP125, boostp250 => BOOSTP250, boostp500 => BOOSTP500,
1042         boostnorm => BOOSTNORM
1043 };
1043 reg(encode, tx_power, Val) ->
1044 % Leave the possibility to the user to write the value as one
1045 case Val of
1046     #{ tx_power := ValToEncode } -> reverse(<<ValToEncode:32>>);
1047     #{ boostp125 := BOOSTP125, boostp250 := BOOSTP250, boostp500 := BOOSTP500,
1048         boostnorm := BOOSTNORM } ->reverse(<<BOOSTP125:8, BOOSTP250:8,
1049         BOOSTP500:8, BOOSTNORM:8>>)
1048 end;
1049 reg(decode, chan_ctrl, Resp) ->
1050 <<
1051     RX_PCODE:5, TX_PCODE:5, RNSSFD:1, TNSSFD:1, RXPRF:2, DWSFD:1, Reserved0:9,
1052     RX_CHAN:4, TX_CHAN:4
1053 >> = reverse(Resp),
1054     #{
1055     rx_pcode => RX_PCODE, tx_pcode => TX_PCODE, rnssfd => RNSSFD, tnssfd =>
1056     TNSSFD, rxprf => RXPRF, dwsfd => DWSFD, res0 => Reserved0, rx_chan =>
1057     RX_CHAN, tx_chan => TX_CHAN
1058 };
1059 reg(encode, chan_ctrl, Val) ->
1060     #{
1061     rx_pcode := RX_PCODE, tx_pcode := TX_PCODE, rnssfd := RNSSFD, tnssfd :=
1062     TNSSFD, rxprf := RXPRF, dwsfd := DWSFD, res0 := Reserved0, rx_chan :=
1063     RX_CHAN, tx_chan := TX_CHAN
1064 } = Val,
1065 reverse(<<
1066     RX_PCODE:5, TX_PCODE:5, RNSSFD:1, TNSSFD:1, RXPRF:2, DWSFD:1, Reserved0:9,
1067     RX_CHAN:4, TX_CHAN:4
1068 >>);
1069 reg(encode, usr_sfd, Value) ->
1070     #{
1071     usr_sfd := USR_SFD
1072 } = Value,
1073 reverse(<<
1074     USR_SFD:(8*41)
1075 >>);
1076 reg(decode, usr_sfd, Resp) ->
1077 <<
1078     USR_SFD:(8*41)
1079 >> = reverse(Resp),
1080     #{
1081     usr_sfd => USR_SFD
1082 };
1083 % AGC_CTRL is a complex register with reserved bits that can't be written
1084 reg(encode, agc_ctrl1, Val) ->
1085     #{
1086     res := Reserved, dis_am := DIS_AM
1087 } = Val,
1088 reverse(<<
1089     Reserved:15, DIS_AM:1
1090 >>);
1091 reg(encode, agc_tune1, Val) ->
1092     #{
1093     agc_tune1 := AGC_TUNE1
1094 } = Val,
1095 reverse(<<
1096     AGC_TUNE1:16
1097 >>);
1098 reg(encode, agc_tune2, Val) ->
1099     #{

```

```

1094     agc_tune2 := AGC_TUNE2
1095     } = Val,
1096     reverse(<<
1097         AGC_TUNE2:32
1098     >>);
1099 reg(encode, agc_tune3, Val) ->
1100     #{
1101         agc_tune3 := AGC_TUNE3
1102     } = Val,
1103     reverse(<<
1104         AGC_TUNE3:16
1105     >>);
1106 reg(decode, agc_ctrl, Resp) ->
1107     <<
1108         _:4, EDV2:9, EDG1:5, _:6, % AGC_STAT1 (RP => don't save reserved bits)
1109         _:80, % Reserved 4
1110         AGC_TUNE3:16, % AGC_TUNE3
1111         _:16, % Reserved 3
1112         AGC_TUNE2:32, % AGC_TUNE2
1113         _:48, % Reserved 2
1114         AGC_TUNE1:16, % AGC_TUNE1
1115         Reserved0:15, DIS_AM:1, % AGC_CTRL1 (RW => save reserved bits)
1116         _:16 % Reserved 1
1117     >> = reverse(Resp),
1118     #{
1119         agc_ctrl1 => #{res => Reserved0, dis_am => DIS_AM},
1120         agc_tune1 => AGC_TUNE1,
1121         agc_tune2 => AGC_TUNE2,
1122         agc_tune3 => AGC_TUNE3,
1123         agc_stat1 => #{edv2 => EDV2, edg1 => EDG1}
1124     };
1125 reg(encode, ec_ctrl, Val) ->
1126     #{
1127         res := Reserved, ostrm := OSTRM, wait := WAIT, pllldt := PLLLDT, osrsm :=
1128             OSRSM, ostsm := OSTSM
1129     } = Val,
1129     reverse(<<
1130         Reserved:20, OSTRM:1, WAIT:8, PLLLDT:1, OSRSM:1, OSTSM:1 % EC_CTRL
1131     >>);
1132 reg(decode, ext_sync, Resp) ->
1133     <<
1134         _:26, OFFSET_EXT:6, % EC_GLOP
1135         RX_TS_EST:32, % EC_RXTC
1136         Reserved:20, OSTRM:1, WAIT:8, PLLLDT:1, OSRSM:1, OSTSM:1 % EC_CTRL
1137     >> = reverse(Resp),
1138     #{
1139         ec_ctrl => #{res => Reserved, ostrm => OSTRM, wait => WAIT, pllldt =>
1140             PLLLDT, osrsm => OSRSM, ostsm => OSTSM},
1141         rx_ts_est => RX_TS_EST,
1142         ec_golp => #{offset_ext => OFFSET_EXT}
1143     };
1143 % "The host system doesn't need to access the ACC_MEM in normal operation, however
1144     it may be of interest [...] for diagnostic purpose" (from DW1000 user manual)
1144 reg(decode, acc_mem, Resp) ->
1145     #{
1146         acc_mem => reverse(Resp)
1147     };
1148 reg(encode, gpio_mode, Val) ->
1149     #{
1150         msgp8 := MSGP8, msgp7 := MSGP7, msgp6 := MSGP6, msgp5 := MSGP5, msgp4 :=
1151             MSGP4, msgp3 := MSGP3, msgp2 := MSGP2, msgp1 := MSGP1, msgp0 := MSGP0
1152     } = Val,

```

```

1152     reverse(<<
1153         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
1154     >>);
1155 reg(encode, gpio_dir, Val) ->
1156     #{
1157         gdm8 := GDM8, gdm7 := GDM7, gdm6 := GDM6, gdm5 := GDM5, gdm4 := GDM4, gdm3
           := GDM3, gdm2 := GDM2, gdm1 := GDM1, gdm0 := GDM0,
1158         gdp8 := GDP8, gdp7 := GDP7, gdp6 := GDP6, gdp5 := GDP5, gdp4 := GDP4, gdp3
           := GDP3, gdp2 := GDP2, gdp1 := GDP1, gdp0 := GDP0
1159     } = Val,
1160     reverse(<<
1161         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
1162     >>);
1163 reg(encode, gpio_dout, Val) ->
1164     #{
1165         gom8 := GOM8, gom7 := GOM7, gom6 := GOM6, gom5 := GOM5, gom4 := GOM4, gom3
           := GOM3, gom2 := GOM2, gom1 := GOM1, gom0 := GOM0,
1166         gop8 := GOP8, gop7 := GOP7, gop6 := GOP6, gop5 := GOP5, gop4 := GOP4, gop3
           := GOP3, gop2 := GOP2, gop1 := GOP1, gop0 := GOP0
1167     } = Val,
1168     reverse(<<
1169         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
1170     >>);
1171 reg(encode, gpio_irqe, Val) ->
1172     #{
1173         girqe8 := GIRQE8, girqe7 := GIRQE7, girqe6 := GIRQE6, girqe5 := GIRQE5,
           girqe4 := GIRQE4, girqe3 := GIRQE3, girqe2 := GIRQE2, girqe1 := GIRQE1
           , girqe0 := GIRQE0
1174     } = Val,
1175     reverse(<<
1176         2#0:23, GIRQE8:1, GIRQE7:1, GIRQE6:1, GIRQE5:1, GIRQE4:1, GIRQE3:1, GIRQE2
           :1, GIRQE1:1, GIRQE0:1 % GPIO_IRQE
1177     >>);
1178 reg(encode, gpio_isen, Val) ->
1179     #{
1180         gisen8 := GISEN8, gisen7 := GISEN7, gisen6 := GISEN6, gisen5 := GISEN5,
           gisen4 := GISEN4, gisen3 := GISEN3, gisen2 := GISEN2, gisen1 := GISEN1
           , gisen0 := GISEN0
1181     } = Val,
1182     reverse(<<
1183         2#0:23, GISEN8:1, GISEN7:1, GISEN6:1, GISEN5:1, GISEN4:1, GISEN3:1, GISEN2
           :1, GISEN1:1, GISEN0:1 % GPIO_ISEN
1184     >>);
1185 reg(encode, gpio_imod, Val) ->
1186     #{
1187         gimod8 := GIMOD8, gimod7 := GIMOD7, gimod6 := GIMOD6, gimod5 := GIMOD5,
           gimod4 := GIMOD4, gimod3 := GIMOD3, gimod2 := GIMOD2, gimod1 := GIMOD1
           , gimod0 := GIMOD0
1188     } = Val,
1189     reverse(<<
1190         2#0:23, GIMOD8:1, GIMOD7:1, GIMOD6:1, GIMOD5:1, GIMOD4:1, GIMOD3:1, GIMOD2
           :1, GIMOD1:1, GIMOD0:1 % GPIO_IMOD
1191     >>);
1192 reg(encode, gpio_ibes, Val) ->
1193     #{
1194         gibes8 := GIBES8, gibes7 := GIBES7, gibes6 := GIBES6, gibes5 := GIBES5,
           gibes4 := GIBES4, gibes3 := GIBES3, gibes2 := GIBES2, gibes1 := GIBES1

```



```

    , gibes0 := GIBES0
1195     } = Val,
1196     reverse(<<
1197         2#0:23, GIBES8:1, GIBES7:1, GIBES6:1, GIBES5:1, GIBES4:1, GIBES3:1, GIBES2
           :1, GIBES1:1, GIBES0:1 % GPIO_IBES
1198     >>);
1199     reg(encode, gpio_iclr, Val) ->
1200     #{
1201         giclr8 := GICLR8, giclr7 := GICLR7, giclr6 := GICLR6, giclr5 := GICLR5,
           giclr4 := GICLR4, giclr3 := GICLR3, giclr2 := GICLR2, giclr1 := GICLR1
           , giclr0 := GICLR0
1202     } = Val,
1203     reverse(<<
1204         2#0:23, GICLR8:1, GICLR7:1, GICLR6:1, GICLR5:1, GICLR4:1, GICLR3:1, GICLR2
           :1, GICLR1:1, GICLR0:1 % GPIO_ICLR
1205     >>);
1206     reg(encode, gpio_idbe, Val) ->
1207     #{
1208         gidbe8 := GIDBE8, gidbe7 := GIDBE7, gidbe6 := GIDBE6, gidbe5 := GIDBE5,
           gidbe4 := GIDBE4, gidbe3 := GIDBE3, gidbe2 := GIDBE2, gidbe1 := GIDBE1
           , gidbe0 := GIDBE0
1209     } = Val,
1210     reverse(<<
1211         2#0:23, GIDBE8:1, GIDBE7:1, GIDBE6:1, GIDBE5:1, GIDBE4:1, GIDBE3:1, GIDBE2
           :1, GIDBE1:1, GIDBE0:1 % GPIO_IDBE
1212     >>);
1213     reg(encode, gpio_raw, Val) ->
1214     #{
1215         grawp8 := GRAWP8, grawp7 := GRAWP7, grawp6 := GRAWP6, grawp5 := GRAWP5,
           grawp4 := GRAWP4, grawp3 := GRAWP3, grawp2 := GRAWP2, grawp1 := GRAWP1
           , grawp0 := GRAWP0
1216     } = Val,
1217     reverse(<<
1218         2#0:23, GRAWP8:1, GRAWP7:1, GRAWP6:1, GRAWP5:1, GRAWP4:1, GRAWP3:1, GRAWP2
           :1, GRAWP1:1, GRAWP0:1 % GPIO_RAW
1219     >>);
1220     reg(decode, gpio_ctrl, Resp) ->
1221     <<
1222     _:23, GRAWP8:1, GRAWP7:1, GRAWP6:1, GRAWP5:1, GRAWP4:1, GRAWP3:1, GRAWP2
           :1, GRAWP1:1, GRAWP0:1, % GPIO_RAW
1223     _:23, GIDBE8:1, GIDBE7:1, GIDBE6:1, GIDBE5:1, GIDBE4:1, GIDBE3:1, GIDBE2
           :1, GIDBE1:1, GIDBE0:1, % GPIO_IDBE
1224     _:23, GICLR8:1, GICLR7:1, GICLR6:1, GICLR5:1, GICLR4:1, GICLR3:1, GICLR2
           :1, GICLR1:1, GICLR0:1, % GPIO_ICLR
1225     _:23, GIBES8:1, GIBES7:1, GIBES6:1, GIBES5:1, GIBES4:1, GIBES3:1, GIBES2
           :1, GIBES1:1, GIBES0:1, % GPIO_IBES
1226     _:23, GIMOD8:1, GIMOD7:1, GIMOD6:1, GIMOD5:1, GIMOD4:1, GIMOD3:1, GIMOD2
           :1, GIMOD1:1, GIMOD0:1, % GPIO_IMOD
1227     _:23, GISEN8:1, GISEN7:1, GISEN6:1, GISEN5:1, GISEN4:1, GISEN3:1, GISEN2
           :1, GISEN1:1, GISEN0:1, % GPIO_ISEN
1228     _:23, GIRQE8:1, GIRQE7:1, GIRQE6:1, GIRQE5:1, GIRQE4:1, GIRQE3:1, GIRQE2
           :1, GIRQE1:1, GIRQE0:1, % GPIO_IRQE
1229     _: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
1230     _: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
1231     _:32, % Reserved
1232     _:8, MSGP8:2, MSGP7:2, MSGP6:2, MSGP5:2, MSGP4:2, MSGP3:2, MSGP2:2, MSGP1
           :2, MSGP0:2, _:6 % GPIO_MODE
1233     >> = reverse(Resp),

```

```

1234     #{
1235         gpio_mode => #{msgp8 => MSGP8, msgp7 => MSGP7, msgp6 => MSGP6, msgp5 =>
            MSGP5, msgp4 => MSGP4, msgp3 => MSGP3, msgp2 => MSGP2, msgp1 => MSGP1,
            msgp0 => MSGP0},
1236         gpio_dir => #{gdm8 => GDM8, gdm7 => GDM7, gdm6 => GDM6, gdm5 => GDM5, gdm4
            => GDM4, gdm3 => GDM3, gdm2 => GDM2, gdm1 => GDM1, gdm0 => GDM0,
1237             gdp8 => GDP8, gdp7 => GDP7, gdp6 => GDP6, gdp5 => GDP5, gdp4
            => GDP4, gdp3 => GDP3, gdp2 => GDP2, gdp1 => GDP1, gdp0
            => GDP0},
1238         gpio_dout => #{gom8 => GOM8, gom7 => GOM7, gom6 => GOM6, gom5 => GOM5,
            gom4 => GOM4, gom3 => GOM3, gom2 => GOM2, gom1 => GOM1, gom0 => GOM0,
1239             gop8 => GOP8, gop7 => GOP7, gop6 => GOP6, gop5 => GOP5,
            gop4 => GOP4, gop3 => GOP3, gop2 => GOP2, gop1 => GOP1,
            gop0 => GOP0},
1240         gpio_irqe => #{girqe8 => GIRQE8, girqe7 => GIRQE7, girqe6 => GIRQE6,
            girqe5 => GIRQE5, girqe4 => GIRQE4, girqe3 => GIRQE3, girqe2 => GIRQE2
            , girqe1 => GIRQE1, girqe0 => GIRQE0},
1241         gpio_isen => #{gisen8 => GISEN8, gisen7 => GISEN7, gisen6 => GISEN6,
            gisen5 => GISEN5, gisen4 => GISEN4, gisen3 => GISEN3, gisen2 => GISEN2
            , gisen1 => GISEN1, gisen0 => GISEN0},
1242         gpio_imod => #{gimod8 => GIMOD8, gimod7 => GIMOD7, gimod6 => GIMOD6,
            gimod5 => GIMOD5, gimod4 => GIMOD4, gimod3 => GIMOD3, gimod2 => GIMOD2
            , gimod1 => GIMOD1, gimod0 => GIMOD0},
1243         gpio_ibes => #{gibes8 => GIBES8, gibes7 => GIBES7, gibes6 => GIBES6,
            gibes5 => GIBES5, gibes4 => GIBES4, gibes3 => GIBES3, gibes2 => GIBES2
            , gibes1 => GIBES1, gibes0 => GIBES0},
1244         gpio_iclr => #{giclr8 => GICLR8, giclr7 => GICLR7, giclr6 => GICLR6,
            giclr5 => GICLR5, giclr4 => GICLR4, giclr3 => GICLR3, giclr2 => GICLR2
            , giclr1 => GICLR1, giclr0 => GICLR0},
1245         gpio_idbe => #{gidbe8 => GIDBE8, gidbe7 => GIDBE7, gidbe6 => GIDBE6,
            gidbe5 => GIDBE5, gidbe4 => GIDBE4, gidbe3 => GIDBE3, gidbe2 => GIDBE2
            , gidbe1 => GIDBE1, gidbe0 => GIDBE0},
1246         gpio_raw => #{grawp8 => GRAWP8, grawp7 => GRAWP7, grawp6 => GRAWP6, grawp5
            => GRAWP5, grawp4 => GRAWP4, grawp3 => GRAWP3, grawp2 => GRAWP2,
            grawp1 => GRAWP1, grawp0 => GRAWP0}
1247     };
1248 reg(encode, drx_tune0b, Val) ->
1249     #{
1250         drx_tune0b := DRX_TUNE0b
1251     } = Val,
1252     reverse(<<
1253         DRX_TUNE0b:16
1254     >>);
1255 reg(encode, drx_tune1a, Val) ->
1256     #{
1257         drx_tune1a := DRX_TUNE1a
1258     } = Val,
1259     reverse(<<
1260         DRX_TUNE1a:16
1261     >>);
1262 reg(encode, drx_tune1b, Val) ->
1263     #{
1264         drx_tune1b := DRX_TUNE1b
1265     } = Val,
1266     reverse(<<
1267         DRX_TUNE1b:16
1268     >>);
1269 reg(encode, drx_tune2, Val) ->
1270     #{
1271         drx_tune2 := DRX_TUNE2
1272     } = Val,
1273     reverse(<<

```

```

1274     DRX_TUNE2:32
1275     >>);
1276 reg(encode, drx_sfdtoc, Val) ->
1277     #{
1278         drx_sfdtoc := DRX_SFDTOC
1279     } = Val,
1280     reverse(<<
1281         DRX_SFDTOC:16
1282     >>);
1283 reg(encode, drx_pretoc, Val) ->
1284     #{
1285         drx_pretoc := DRX_PRETOC
1286     } = Val,
1287     reverse(<<
1288         DRX_PRETOC:16
1289     >>);
1290 reg(encode, drx_tune4h, Val) ->
1291     #{
1292         drx_tune4h := DRX_TUNE4H
1293     } = Val,
1294     reverse(<<
1295         DRX_TUNE4H:16
1296     >>);
1297 reg(decode, drx_conf, Resp) ->
1298     <<
1299         RXPACC_NOSAT:8, % present in the user manual but not in the driver code in
1300             C
1301         % _:8, % Placeholder for the remaining 8 bits
1302         DRX_CAR_INT:24,
1303         DRX_TUNE4H:16,
1304         DRX_PRETOC:16,
1305         _:16,
1306         DRX_SFDTOC:16,
1307         _:160,
1308         DRX_TUNE2:32,
1309         DRX_TUNE1b:16,
1310         DRX_TUNE1a:16,
1311         DRX_TUNE0b:16,
1312         _:16
1313     >> = reverse(Resp),
1314     #{
1315         drx_tune0b => DRX_TUNE0b,
1316         drx_tune1a => DRX_TUNE1a,
1317         drx_tune1b => DRX_TUNE1b,
1318         drx_tune2 => DRX_TUNE2,
1319         drx_tune4h => DRX_TUNE4H,
1320         drx_car_int => DRX_CAR_INT,
1321         drx_sfdtoc => DRX_SFDTOC,
1322         drx_pretoc => DRX_PRETOC,
1323         rxpacc_nosat => RXPACC_NOSAT
1324     };
1325 reg(encode, rf_conf, Val) ->
1326     #{
1327         txrxsw := TXRXSW, ldofen := LDOFEN, pllflen := PLLFEN, txfen := TXFEN
1328     } = Val,
1329     reverse(<<
1330         2#0:9, TXRXSW:2, LDOFEN:5, PLLFEN:3, TXFEN:5, 2#0:8 % RF_CONF
1331     >>);
1332 reg(encode, rf_rxctrlh, Val) ->
1333     #{
1334         rf_rxctrlh := RF_RXCTRLH
1335     } = Val,

```

```

1335     reverse(<<
1336         RF_RXCTRLH:8 % RF_RXCTRLH
1337     >>);
1338 % user manual gives fields but encoding should be done as one following table 38
1339 reg(encode, rf_txctrl, Val) ->
1340     #{
1341         rf_txctrl := RF_TXCTRL
1342     } = Val,
1343     reverse(<<
1344         RF_TXCTRL:32
1345     >>);
1346 reg(encode, ldotune, Val) ->
1347     #{
1348         ldotune := LDOTUNE
1349     } = Val,
1350     reverse(<<
1351         LDOTUNE:40
1352     >>);
1353 reg(decode, rf_conf, Resp) ->
1354     <<
1355         _:40, % Placeholder for the remaining 40 bits
1356         LDOTUNE:40, % LDOTUNE
1357         _:28, RFPLLLOCK:1, CPLLHIGH:1, CPLLLOW:1, CPLLLOCK:1, % RF_STATUS
1358         _:128, _:96, % Reserved 2 - On user manual 16 bytes but offset gives 28
1359         RF_TXCTRL:32, % cf. encode function: Reserved:20, TXMQ:3, TXMTUNE:4, _:5 -
1360         RF_TXCTRL
1361         RF_RXCTRLH:8, % RF_RXCTRLH
1362         _:56, % Reserved 1
1363         _:9, TXRXSW:2, LDOFEN:5, PLLFEN:3, TXFEN:5, _:8 % RF_CONF
1364     >> = reverse(Resp),
1365     #{
1366         ldotune => LDOTUNE,
1367         rf_status => #{rfplllock => RFPLLLOCK, cpllhigh =>
1368         CPLLHIGH, cplllock => CPLLLOCK},
1369         rf_txctrl => RF_TXCTRL,
1370         rf_rxctrlh => RF_RXCTRLH,
1371         rf_conf => #{txrxsw => TXRXSW, ldofen => LDOFEN, pllfen => PLLFEN, txfen
1372         => TXFEN}
1373     };
1374 reg(encode, tc_sarc, Val) ->
1375     #{
1376         sar_ctrl := SAR_CTRL
1377     } = Val,
1378     reverse(<<
1379         2#0:15, SAR_CTRL:1
1380     >>);
1381 reg(encode, tc_pg_ctrl, Val) ->
1382     #{
1383         pg_tmeas := PG_TMEAS, res := Reserved, pg_start := PG_START
1384     } = Val,
1385     reverse(<<
1386         2#0:2, PG_TMEAS:4, Reserved:1, PG_START:1
1387     >>);
1388 reg(encode, tc_pgdelay, Val) ->
1389     #{
1390         tc_pgdelay := TC_PGDELAY
1391     } = Val,
1392     reverse(<<
1393         TC_PGDELAY:8
1394     >>);
1395 reg(encode, tc_pgtest, Val) ->

```

```

1393     #{
1394         tc_pgctest := TC_PGTEST
1395     } = Val,
1396     reverse(<<
1397         TC_PGTEST:8
1398     >>);
1399 reg(decode, tx_cal, Resp) ->
1400 <<
1401     TC_PGTEST:8, % TC_PGTEST
1402     TC_PGDELAY:8, % TC_PGDELAY
1403     _:4, DELAY_CNT:12, % TC_PG_STATUS
1404     _:2, PG_TMEAS:4, Reserved0:1, PG_START:1, % TC_PG_CTRL
1405     SAR_WTEMP:8, SAR_WVBAT:8, % TC_SARW
1406     _:8, SAR_LTEMP:8, SAR_LVBAT:8, % TC_SARL
1407     _:8, % Place holder to fill the gap between the offsets
1408     _:15, SAR_CTRL:1 % TC_SARC
1409 >> = reverse(Resp),
1410     #{
1411         tc_pgctest => TC_PGTEST,
1412         tc_pgdelay => TC_PGDELAY,
1413         tc_pg_status => #{delay_cnt => DELAY_CNT},
1414         tc_pg_ctrl => #{pg_tmeas => PG_TMEAS, res => Reserved0, pg_start =>
1415             PG_START},
1416         tc_sarw => #{sar_wtemp => SAR_WTEMP, sar_wvbat => SAR_WVBAT},
1417         tc_sarl => #{sar_ltemp => SAR_LTEMP, sar_lvbat => SAR_LVBAT},
1418         tc_sarc => #{sar_ctrl => SAR_CTRL}
1419     };
1419 reg(encode, fs_pllcfg, Val) ->
1420     #{
1421         fs_pllcfg := FS_PLLCFG
1422     } = Val,
1423     reverse(<<
1424         FS_PLLCFG:32
1425     >>);
1426 reg(encode, fs_plltune, Val) ->
1427     #{
1428         fs_plltune := FS_PLLTUNE
1429     } = Val,
1430     reverse(<<
1431         FS_PLLTUNE:8
1432     >>);
1433 reg(encode, fs_xtalt, Val) ->
1434     #{
1435         res := Reserved, xtalt := XTALT
1436     } = Val,
1437     reverse(<<
1438         Reserved:3, XTALT:5
1439     >>);
1440 reg(decode, fs_ctrl, Resp) ->
1441 <<
1442     _:48, % Reserved 3
1443     Reserved:3, XTALT:5, % FS_XTALT
1444     _:16, % Reserved 2
1445     FS_PLLTUNE:8, % FS_PLLTUNE
1446     FS_PLLCFG:32, % FS_PLLCFG
1447     _:56 % Reserved 1
1448 >> = reverse(Resp),
1449     #{
1450         fs_xtalt => #{res => Reserved, xtalt => XTALT},
1451         fs_plltune => FS_PLLTUNE,
1452         fs_pllcfg => FS_PLLCFG
1453     };

```

```

1454 reg(encode, aon_wcfg, Val) ->
1455     #{
1456         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
1457     } = Val,
1458     reverse(<<
1459         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
1460     >>);
1461 reg(encode, aon_ctrl, Val) ->
1462     #{
1463         dca_enab := DCA_ENAB, dca_read := DCA_READ, upl_cfg := UPL_CFG, save :=
           SAVE, restore := RESTORE
1464     } = Val,
1465     reverse(<<
1466         DCA_ENAB:1, 2#0:3, DCA_READ:1, UPL_CFG:1, SAVE:1, RESTORE:1 % AON_CTRL
1467     >>);
1468 reg(encode, aon_rdat, Val) ->
1469     #{
1470         aon_rdat := AON_RDAT
1471     } = Val,
1472     reverse(<<
1473         AON_RDAT:8 % AON_RDAT
1474     >>);
1475 reg(encode, aon_addr, Val) ->
1476     #{
1477         aon_addr := AON_ADDR
1478     } = Val,
1479     reverse(<<
1480         AON_ADDR:8 % AON_ADDR
1481     >>);
1482 reg(encode, aon_cfg0, Val) ->
1483     #{
1484         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
1485     } = Val,
1486     reverse(<<
1487         SLEEP_TIM:16, LPCLKDIVA:11, LPDIV_EN:1, WAKE_CNT:1, WAKE_SPI:1, WAKE_PIN
           :1, SLEEP_EN:1 % AON_CFG0
1488     >>);
1489 reg(encode, aon_cfg1, Val) ->
1490     #{
1491         res := Reserved, lposc_c := LPOSC_C, smxx := SMXX, sleep_ce := SLEEP_CE
1492     } = Val,
1493     reverse(<<
1494         Reserved:13, LPOSC_C:1, SMXX:1, SLEEP_CE:1 % AON_CFG1
1495     >>);
1496 reg(decode, aon, Resp) ->
1497     <<
1498         Reserved:13, LPOSC_C:1, SMXX:1, SLEEP_CE:1, % AON_CFG1
1499         SLEEP_TIM:16, LPCLKDIVA:11, LPDIV_EN:1, WAKE_CNT:1, WAKE_SPI:1, WAKE_PIN
           :1, SLEEP_EN:1, % AON_CFG0
1500         _:8, % Reserved 1
1501         AON_ADDR:8, % AON_ADDR
1502         AON_RDAT:8, % AON_RDAT
1503         DCA_ENAB:1, _:3, DCA_READ:1, UPL_CFG:1, SAVE:1, RESTORE:1, % AON_CTRL
1504         _: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
1505     >> = reverse(Resp),
1506     #{

```

```

1507     aon_cfg1 => #{res => Reserved, lposc_c => LPOSC_C, smxx => SMXX, sleep_ce
=> SLEEP_CE},
1508     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},
1509     aon_addr => AON_ADDR,
1510     aon_rdat => AON_RDAT,
1511     aon_ctrl => #{dca_enab => DCA_ENAB, dca_read => DCA_READ, upl_cfg =>
UPL_CFG, save => SAVE, restore => RESTORE},
1512     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}
1513 };
1514 reg(encode, otp_wdat, Val) ->
1515   #{
1516     otp_wdat := OTP_WDAT
1517   } = Val,
1518   reverse(<<
1519     OTP_WDAT:32 % OTP_WDAT
1520   >>);
1521 reg(encode, otp_addr, Val) ->
1522   #{
1523     otpaddr := OTP_ADDR, res := Reserved
1524   } = Val,
1525   reverse(<<
1526     Reserved:5, OTP_ADDR:11 % OTP_ADDR
1527   >>);
1528 reg(encode, otp_ctrl, Val) ->
1529   #{
1530     ldeload := LDELOAD, res1 := Reserved1, otpmr := OTPMR, otpprog := OTPPROG,
res2 := Reserved2, otpmrwr := OTPMRWR, res3 := Reserved3, otpread :=
OTPREAD, otp_rden := OTPRDEN
1531   } = Val,
1532   reverse(<<
1533     LDELOAD:1, Reserved1:4, OTPMR:4, OTPPROG:1, Reserved2:2, OTPMRWR:1,
Reserved3:1, OTPREAD:1, OTPRDEN:1 % OTP_CTRL
1534   >>);
1535 reg(encode, otp_stat, Val) ->
1536   #{
1537     res := Reserved, otp_vpok := OTP_VPOK, otpprgd := OTPPRGD
1538   } = Val,
1539   reverse(<<
1540     Reserved:14, OTP_VPOK:1, OTPPRGD:1 % OTP_STAT
1541   >>);
1542 reg(encode, otp_rdat, Val) ->
1543   #{
1544     otp_rdat := OTP_RDAT
1545   } = Val,
1546   reverse(<<
1547     OTP_RDAT:32 % OTP_RDAT
1548   >>);
1549 reg(encode, opt_srdat, Val) ->
1550   #{
1551     otp_srdat := OTP_SRDAT
1552   } = Val,
1553   reverse(<<
1554     OTP_SRDAT:32 % OTP_SRDAT
1555   >>);
1556 reg(encode, otp_sf, Val) ->
1557   #{
1558     res1 := Reserved1, ops_sel := OPS_SEL, res2 := Reserved2, ldo_kick :=
LDO_KICK, ops_kick := OPS_KICK

```

```

1559     } = Val,
1560     reverse(<<
1561         Reserved1:2, OPS_SEL:1, Reserved2:3, LDO_KICK:1, OPS_KICK:1 % OTP_SF
1562     >>);
1563 reg(decode, otp_if, Resp) ->
1564     <<
1565         Reserved5:2, OPS_SEL:1, Reserved6:3, LDO_KICK:1, OPS_KICK:1, % OTP_SF
1566         OTP_SRDAT:32, % OTP_SRDAT
1567         OTP_RDAT:32, % OTP_RDAT
1568         Reserved4:14, OTP_VPOK:1, OTPPRGD:1, % OTP_STAT
1569         LDELOAD:1, Reserved1:4, OTPMR:4, OTPPROG:1, Reserved2:2, OTPMRWR:1,
1570         Reserved3:1, OTPREAD:1, OTPRDEN:1, % OTP_CTRL
1571         Reserved0:5, OTP_ADDR:11, % OTP_ADDR
1572         OTP_WDAT:32 % OTP_WDAT
1573     >> = reverse(Resp),
1574     #{
1575         otp_sf => #{res1 => Reserved5, ops_sel => OPS_SEL, res2 => Reserved6,
1576             ldo_kick => LDO_KICK, ops_kick => OPS_KICK},
1577         otp_srdat => OTP_SRDAT,
1578         otp_rdat => OTP_RDAT,
1579         otp_stat => #{res => Reserved4, otp_vpok => OTP_VPOK, otpprgd => OTPPRGD},
1580         otp_ctrl => #{lde_load => LDELOAD, res1 => Reserved1, otpmr => OTPMR,
1581             otpprog => OTPPROG, res2 => Reserved2, otpmrwr => OTPMRWR, res3 =>
1582             Reserved3, otpread => OTPREAD, otp_rden => OTPRDEN},
1583         otp_addr => #{otpaddr => OTP_ADDR, res => Reserved0},
1584         otp_wdat => OTP_WDAT
1585     };
1586 reg(decode, lde_thresh, Resp) ->
1587     <<
1588         LDE_THRESH:16
1589     >> = reverse(Resp),
1590     #{
1591         lde_thresh => LDE_THRESH
1592     };
1593 reg(encode, lde_cfg1, Val) ->
1594     #{
1595         pmult := PMULT, ntm := NTM
1596     } = Val,
1597     reverse(<<
1598         PMULT:3, NTM:5
1599     >>);
1600 reg(decode, lde_cfg1, Resp) ->
1601     <<
1602         PMULT:3, NTM:5
1603     >> = reverse(Resp),
1604     #{
1605         lde_cfg1 => #{pmult => PMULT, ntm => NTM}
1606     };
1607 reg(decode, lde_ppindx, Resp) ->
1608     <<
1609         LDE_PPINDX:16
1610     >> = reverse(Resp),
1611     #{
1612         lde_ppindx => LDE_PPINDX
1613     };
1614 reg(decode, lde_ppampl, Resp) ->
1615     <<
1616         LDE_PPAMPL:16
1617     >> = reverse(Resp),
1618     #{
1619         lde_ppampl => LDE_PPAMPL
1620     };

```



```

1617 reg(encode, lde_rxantd, Val) ->
1618     #{
1619         lde_rxantd := LDE_RXANTD
1620     } = Val,
1621     reverse(<<
1622         LDE_RXANTD:16
1623     >>);
1624 reg(decode, lde_rxantd, Resp) ->
1625     <<
1626         LDE_RXANTD:16
1627     >> = reverse(Resp),
1628     #{
1629         lde_rxantd => LDE_RXANTD
1630     };
1631 reg(encode, lde_cfg2, Val) ->
1632     #{
1633         lde_cfg2 := LDE_CFG2
1634     } = Val,
1635     reverse(<<
1636         LDE_CFG2:16
1637     >>);
1638 reg(decode, lde_cfg2, Resp) ->
1639     <<
1640         LDE_CFG2:16
1641     >> = reverse(Resp),
1642     #{
1643         lde_cfg2 => LDE_CFG2
1644     };
1645 reg(encode, lde_repc, Val) ->
1646     #{
1647         lde_repc := LDE_REPC
1648     } = Val,
1649     reverse(<<
1650         LDE_REPC:16
1651     >>);
1652 reg(decode, lde_repc, Resp) ->
1653     <<
1654         LDE_REPC:16
1655     >> = reverse(Resp),
1656     #{
1657         lde_repc => LDE_REPC
1658     };
1659 reg(encode, evc_ctrl, Val) ->
1660     #{
1661         evc_clr := EVC_CLR, evc_en := EVC_EN
1662     } = Val,
1663     reverse(<<
1664         2#0:30, EVC_CLR:1, EVC_EN:1 % EVC_CTRL
1665     >>);
1666 reg(encode, diag_tmc, Val) ->
1667     #{
1668         tx_pstm := TX_PSTM
1669     } = Val,
1670     reverse(<<
1671         2#0:11, TX_PSTM:1, 2#0:4 % DIAG_TMC
1672     >>);
1673 reg(decode, dig_diag, Resp) ->
1674     <<
1675         _:11, TX_PSTM:1, _:4, % DIAG_TMC
1676         _:64, % Reserved 1
1677         _:4, EVC_TPW:12, % EVC_TPW
1678         _:4, EVC_HPW:12, % EVC_HPW

```

```

1679     _:4, EVC_TXFS:12, % EVC_TXFS
1680     _:4, EVC_FWTO:12, % EVC_FWTO
1681     _:4, EVC_PTO:12, % EVC_PTO
1682     _:4, EVC_STO:12, % EVC_STO
1683     _:4, ECV_OVR:12, % EVC_OVR
1684     _:4, EVC_FFR:12, % EVC_FFR
1685     _:4, EVC_FCE:12, % EVC_FCE
1686     _:4, EVC_FCG:12, % EVC_FCG
1687     _:4, EVC_RSE:12, % EVC_RSE
1688     _:4, EVC_PHE:12, % EVC_PHE
1689     _:30, EVC_CLR:1, EVC_EN:1 % EVC_CTRL
1690 >> = reverse(Resp),
1691 #{
1692     diag_tmc => #{tx_pstm => TX_PSTM},
1693     evc_tpw => EVC_TPW,
1694     evc_hpw => EVC_HPW,
1695     evc_txfs => EVC_TXFS,
1696     evc_fwto => EVC_FWTO,
1697     evc_pto => EVC_PTO,
1698     evc_sto => EVC_STO,
1699     evc_ovr => ECV_OVR,
1700     evc_ffr => EVC_FFR,
1701     evc_fce => EVC_FCE,
1702     evc_fcg => EVC_FCG,
1703     evc_rse => EVC_RSE,
1704     evc_phe => EVC_PHE,
1705     evc_ctrl => #{evc_clr => EVC_CLR, evc_en => EVC_EN}
1706 };
1707 reg(encode, pmsc_ctrl0, Val) ->
1708 #{
1709     softreset := SOFTRESET, pll2_seq_en := PLL2_SEQ_EN, khzclken := KHZCLKEN,
1710     gpdrn := GPDRN, gpdcce := GPDCE,
1711     gprn := GPRN, gpce := GPCE, amce := AMCE, adcce := ADCCE, otp := OTP, res8
1712     := Res8, res7 := Res7, face := FACE, txclks := TXCLKS, rxclks :=
1713     RXCLKS, sysclks := SYSCLKS % Here we need res8 for the initial config
1714     of the DW1000. We need to write it
1715 } = Val,
1716 reverse(<<
1717     SOFTRESET:4, 2#000:3, PLL2_SEQ_EN:1, KHZCLKEN:1, 2#011:3, GPDRN:1, GPDCE
1718     :1, GPRN:1, GPCE:1, AMCE:1, 2#000:4, ADCCE:1, OTP:1, Res8:1, Res7:1,
1719     FACE:1, TXCLKS:2, RXCLKS:2, SYSCLKS:2 % PMSC_CTRL0
1720 >>);
1721 reg(encode, pmsc_ctrl1, Val) ->
1722 #{
1723     khzclkdiv := KHZCLKDIV, lderune := LDERUNE, pllsyn := PLLSYN, snozr :=
1724     SNOZR, snoze := SNOZE, arxslp := ARXSLP, atxslp := ATXSLP, pktseq :=
1725     PKTSEQ, arx2init := ARX2INIT
1726 } = Val,
1727 reverse(<<
1728     KHZCLKDIV:6, 2#01000000:8, LDERUNE:1, 2#0:1, PLLSYN:1, SNOZR:1, SNOZE:1,
1729     ARXSLP:1, ATXSLP:1, PKTSEQ:8, 2#0:1, ARX2INIT:1, 2#0:1 % PMSC_CTRL1
1730 >>);
1731 reg(encode, pmsc_snozt, Val) ->
1732 #{
1733     snoz_tim := SNOZ_TIM
1734 } = Val,
1735 reverse(<<
1736     SNOZ_TIM:8 % PMSC_SNOZT
1737 >>);
1738 reg(encode, pmsc_txfseq, Val) ->
1739 #{
1740     txfineseq := TXFINESEQ

```

```

1732     } = Val,
1733     reverse(<<
1734         TXFINESEQ:16 % PMSC_TXFINESEQ
1735     >>);
1736 reg(encode, pmsc_ledc, Val) ->
1737     #{
1738         res31 := RES31, blnknow := BLNKNOW, res15 := RES15, blnken := BLNKEN,
1739         blink_tim := BLINK_TIM
1740     } = Val,
1741     reverse(<<
1742         RES31:12, BLNKNOW:4, RES15:7, BLNKEN:1, BLINK_TIM:8 % PMSC_LEDC
1743     >>);
1744 % mapping pmsc ctrl0 from: https://forum.qorvo.com/t/pmsc-ctrl0-bits8-15/746/3
1745 reg(decode, pmsc, Resp) ->
1746     % User manual says: reserved bits should be preserved at their reset value =>
1747     % can hardcode their values ? Safe to do that ?
1748     <<
1749         Res31:12, BLNKNOW:4, Res15:7, BLNKEN:1, BLINK_TIM:8, % PMSC_LEDC
1750         TXFINESEQ:16, % PMSC_TXFINESEQ
1751         _:(25*8), % Reserved 2
1752         SNOZ_TIM:8, % PMSC_SNOZT
1753         _:32, % Reserved 1
1754         KHZCLKDIV:6, _:8, LDERUNE:1, _:1, PLLSYN:1, SNOZR:1, SNOZE:1, ARXSLP:1,
1755         ATXSLP:1, PKTSEQ:8, _:1, ARX2INIT:1, _:1, % PMSC_CTRL1
1756         SOFTRESET:4, _:3, PLL2_SEQ_EN:1, KHZCLKEN:1, _:3, GPDRN:1, GPDCE:1, GPRN
1757         :1, GPCE:1, AMCE:1, _:4, ADCCE:1, OTP:1, Res8:1, Res7:1, FACE:1,
1758         TXCLKS:2, RXCLKS:2, SYSCLKS:2 % PMSC_CTRL0
1759     >> = reverse(Resp),
1760     #{
1761         pmsc_ledc => #{res31 => Res31, blnknow => BLNKNOW, res15 => Res15, blnken
1762             => BLNKEN, blink_tim => BLINK_TIM},
1763         pmsc_txseq => #{txfineseq => TXFINESEQ},
1764         pmsc_snozt => #{snoz_tim => SNOZ_TIM},
1765         pmsc_ctrl1 => #{khzclkdiv => KHZCLKDIV, lderune => LDERUNE, pllsyn =>
1766             PLLSYN, snozr => SNOZR, snoze => SNOZE, arxslp => ARXSLP, atxslp =>
1767             ATXSLP, pktseq => PKTSEQ, arx2init => ARX2INIT},
1768         pmsc_ctrl0 => #{softreset => SOFTRESET, pll2_seq_en => PLL2_SEQ_EN,
1769             khzclken => KHZCLKEN, gpdrn => GPDRN, gpdce => GPDCE, gprn => GPRN,
1770             gpce => GPCE, amce => AMCE, adcce => ADCCE, otp => OTP, res8 => Res8,
1771             res7 => Res7, face => FACE, txclks => TXCLKS, rxclks => RXCLKS,
1772             sysclks => SYSCLKS}
1773     };
1774 reg(decode, RegFile, Resp) -> error({unknown_regfile_to_decode, RegFile, Resp});
1775 reg(encode, RegFile, Resp) -> error({unknown_regfile_to_encode, RegFile, Resp}).
1776
1777 rw(read) -> 0;
1778 rw(write) -> 1.
1779
1780 % Mapping of the different register IDs to their hexadecimal value
1781 regFile(dev_id) -> 16#00;
1782 regFile(eui) -> 16#01;
1783 % 0x02 is reserved
1784 regFile(panadr) -> 16#03;
1785 regFile(sys_cfg) -> 16#04;
1786 % 0x05 is reserved
1787 regFile(sys_time) -> 16#06;
1788 % 0x07 is reserved
1789 regFile(tx_fctrl) -> 16#08;
1790 regFile(tx_buffer) -> 16#09;
1791 regFile(dx_time) -> 16#0A;
1792 % 0x0B is reserved
1793 regFile(rx_fwto) -> 16#0C;

```

```

1782 regFile(sys_ctrl) -> 16#0D;
1783 regFile(sys_mask) -> 16#0E;
1784 regFile(sys_status) -> 16#0F;
1785 regFile(rx_finfo) -> 16#10;
1786 regFile(rx_buffer) -> 16#11;
1787 regFile(rx_fqual) -> 16#12;
1788 regFile(rx_ttcki) -> 16#13;
1789 regFile(rx_ttcko) -> 16#14;
1790 regFile(rx_time) -> 16#15;
1791 % 0x16 is reserved
1792 regFile(tx_time) -> 16#17;
1793 regFile(tx_antd) -> 16#18;
1794 regFile(sys_state) -> 16#19;
1795 regFile(ack_resp_t) -> 16#1A;
1796 % 0x1B is reserved
1797 % 0x1C is reserved
1798 regFile(rx_sniff) -> 16#1D;
1799 regFile(tx_power) -> 16#1E;
1800 regFile(chan_ctrl) -> 16#1F;
1801 % 0x20 is reserved
1802 regFile(usr_sfd) -> 16#21;
1803 % 0x22 is reserved
1804 regFile(agc_ctrl) -> 16#23;
1805 regFile(ext_sync) -> 16#24;
1806 regFile(acc_mem) -> 16#25;
1807 regFile(gpio_ctrl) -> 16#26;
1808 regFile(drx_conf) -> 16#27;
1809 regFile(rf_conf) -> 16#28;
1810 % 0x29 is reserved
1811 regFile(tx_cal) -> 16#2A;
1812 regFile(fs_ctrl) -> 16#2B;
1813 regFile(aon) -> 16#2C;
1814 regFile(otp_if) -> 16#2D;
1815 regFile(lde_ctrl) -> regFile(lde_if); % No size ?
1816 regFile(lde_if) -> 16#2E;
1817 regFile(dig_diag) -> 16#2F;
1818 % 0x30 - 0x35 are reserved
1819 regFile(pmsc) -> 16#36;
1820 % 0x37 - 0x3F are reserved
1821 regFile(RegId) -> error({wrong_register_ID, RegId}).
1822
1823 % Only the writable subregisters in SRW register files are present here
1824 % AGC_CTRL
1825 subReg(agc_ctrl1) -> 16#02;
1826 subReg(agc_tune1) -> 16#04;
1827 subReg(agc_tune2) -> 16#0C;
1828 subReg(agc_tune3) -> 16#12;
1829 subReg(agc_stat1) -> 16#1E;
1830 subReg(ec_ctrl) -> 16#00;
1831 subReg(gpio_mode) -> 16#00;
1832 subReg(gpio_dir) -> 16#08;
1833 subReg(gpio_dout) -> 16#0C;
1834 subReg(gpio_irqe) -> 16#10;
1835 subReg(gpio_isen) -> 16#14;
1836 subReg(gpio_imode) -> 16#18;
1837 subReg(gpio_ibes) -> 16#1C;
1838 subReg(gpio_iclr) -> 16#20;
1839 subReg(gpio_idbe) -> 16#24;
1840 subReg(gpio_raw) -> 16#28;
1841 subReg(drx_tune0b) -> 16#02;
1842 subReg(drx_tune1a) -> 16#04;
1843 subReg(drx_tune1b) -> 16#06;

```

```

1844 subReg(drx_tune2) -> 16#08;
1845 subReg(drx_sfdtoc) -> 16#20;
1846 subReg(drx_pretoc) -> 16#24;
1847 subReg(drx_tune4h) -> 16#26;
1848 subReg(rf_conf) -> 16#00;
1849 subReg(rf_rxctrlh) -> 16#0B;
1850 subReg(rf_txctrl) -> 16#0C;
1851 subReg(ldotune) -> 16#30;
1852 subReg(tc_sarc) -> 16#00;
1853 subReg(tc_pg_ctrl) -> 16#08;
1854 subReg(tc_pgdelay) -> 16#0B;
1855 subReg(tc_pgtest) -> 16#0C;
1856 subReg(fs_pllcfg) -> 16#07;
1857 subReg(fs_plltune) -> 16#0B;
1858 subReg(fs_xtalt) -> 16#0E;
1859 subReg(aon_wcfg) -> 16#00;
1860 subReg(aon_ctrl) -> 16#02;
1861 subReg(aon_rdat) -> 16#03;
1862 subReg(aon_addr) -> 16#04;
1863 subReg(aon_cfg0) -> 16#06;
1864 subReg(aon_cfg1) -> 16#0A;
1865 subReg(otp_wdat) -> 16#00;
1866 subReg(otp_addr) -> 16#04;
1867 subReg(otp_ctrl) -> 16#06;
1868 subReg(otp_stat) -> 16#08;
1869 subReg(otp_rdat) -> 16#0A;
1870 subReg(otp_srdat) -> 16#0E;
1871 subReg(otp_sf) -> 16#12;
1872 subReg(lde_thresh) -> 16#00;
1873 subReg(lde_cfg1) -> 16#806;
1874 subReg(lde_ppindx) -> 16#1000;
1875 subReg(lde_ppampl) -> 16#1002;
1876 subReg(lde_rxantd) -> 16#1804;
1877 subReg(lde_cfg2) -> 16#1806;
1878 subReg(lde_repc) -> 16#2804;
1879 subReg(evc_ctrl) -> 16#00;
1880 subReg(diag_tmc) -> 16#24;
1881 subReg(pmsc_ctrl0) -> 16#00;
1882 subReg(pmsc_ctrl1) -> 16#04;
1883 subReg(pmsc_snozt) -> 16#0C;
1884 subReg(pmsc_txfseq) -> 16#26;
1885 subReg(pmsc_ledc) -> 16#28.
1886
1887
1888 % Mapping of the size in bytes of the different register IDs
1889 regSize(dev_id) -> 4;
1890 regSize(eui) -> 8;
1891 regSize(panadr) -> 4;
1892 regSize(sys_cfg) -> 4;
1893 regSize(sys_time) -> 5;
1894 regSize(tx_fctrl) -> 5;
1895 regSize(tx_buffer) -> 1024;
1896 regSize(dx_time) -> 5;
1897 regSize(rx_fwto) -> 2; % user manual gives 2 bytes and bits 16-31 are reserved
1898 regSize(sys_ctrl) -> 4;
1899 regSize(sys_mask) -> 4;
1900 regSize(sys_status) -> 5;
1901 regSize(rx_finfo) -> 4;
1902 regSize(rx_buffer) -> 1024;
1903 regSize(rx_fqual) -> 8;
1904 regSize(rx_ttcki) -> 4;
1905 regSize(rx_ttcko) -> 5;

```

```

1906 regSize(rx_time) -> 14;
1907 regSize(tx_time) -> 10;
1908 regSize(tx_antd) -> 2;
1909 regSize(sys_state) -> 4;
1910 regSize(ack_resp_t) -> 4;
1911 regSize(rx_sniff) -> 4;
1912 regSize(tx_power) -> 4;
1913 regSize(chan_ctrl) -> 4;
1914 regSize(usr_sfd) -> 41;
1915 regSize(agc_ctrl) -> 33;
1916 regSize(ext_sync) -> 12;
1917 regSize(acc_mem) -> 4064;
1918 regSize(gpio_ctrl) -> 44;
1919 regSize(drx_conf) -> 44; % user manual gives 44 bytes but sum of register length
    gives 45 bytes
1920 regSize(rf_conf) -> 58; % user manual gives 58 but sum of all its register gives
    53 => Placeholder for the remaining 8 bytes
1921 regSize(tx_cal) -> 13; % user manual gives 52 bytes but sum of all sub regs gives
    13 bytes
1922 regSize(fs_ctrl) -> 21;
1923 regSize(aon) -> 12;
1924 regSize(otp_if) -> 19; % user manual gives 18 bytes in regs table but sum of all
    sub regs is 19 bytes
1925 regSize(lde_ctrl) -> undefined; % No size ?
1926 regSize(lde_if) -> undefined; % No size ?
1927 regSize(dig_diag) -> 38; % user manual gives 41 bytes but sum of all sub regs
    gives 38 bytes
1928 regSize(pm_sc) -> 44. % user manual gives 48 bytes but sum of all sub regs gives 41
    bytes
1929
1930 %% Gives the size in bytes
1931 subRegSize(agc_ctrl1) -> 2;
1932 subRegSize(agc_tune1) -> 2;
1933 subRegSize(agc_tune2) -> 4;
1934 subRegSize(agc_tune3) -> 2;
1935 subRegSize(agc_stat1) -> 3;
1936 subRegSize(ec_ctrl) -> 4;
1937 subRegSize(gpio_mode) -> 4;
1938 subRegSize(gpio_dir) -> 4;
1939 subRegSize(gpio_dout) -> 4;
1940 subRegSize(gpio_irqe) -> 4;
1941 subRegSize(gpio_isen) -> 4;
1942 subRegSize(gpio_imode) -> 4;
1943 subRegSize(gpio_ibes) -> 4;
1944 subRegSize(gpio_iclr) -> 4;
1945 subRegSize(gpio_idbe) -> 4;
1946 subRegSize(gpio_raw) -> 4;
1947 subRegSize(drx_tune0b) -> 2;
1948 subRegSize(drx_tune1a) -> 2;
1949 subRegSize(drx_tune1b) -> 2;
1950 subRegSize(drx_tune2) -> 4;
1951 subRegSize(drx_sfdtoc) -> 2;
1952 subRegSize(drx_pretoc) -> 2;
1953 subRegSize(drx_tune4h) -> 2;
1954 subRegSize(rf_conf) -> 4;
1955 subRegSize(rf_rxctrlh) -> 1;
1956 subRegSize(rf_txctrl) -> 4; % ! table in user manual gives 3 but details gives 4
1957 subRegSize(ldotune) -> 5;
1958 subRegSize(tc_sarc) -> 2;
1959 subRegSize(tc_pg_ctrl) -> 1;
1960 subRegSize(tc_pgdelay) -> 1;
1961 subRegSize(tc_pgtest) -> 1;

```

```

1962 subRegSize(fs_pllcfg) -> 4;
1963 subRegSize(fs_plltune) -> 1;
1964 subRegSize(fs_xtalt) -> 1;
1965 subRegSize(aon_wcfg) -> 2;
1966 subRegSize(aon_ctrl) -> 1;
1967 subRegSize(aon_rdat) -> 1;
1968 subRegSize(aon_addr) -> 1;
1969 subRegSize(aon_cfg0) -> 4;
1970 subRegSize(aon_cfg1) -> 2;
1971 subRegSize(otp_wdat) -> 4;
1972 subRegSize(otp_addr) -> 2;
1973 subRegSize(otp_ctrl) -> 2;
1974 subRegSize(otp_stat) -> 2;
1975 subRegSize(otp_rdat) -> 4;
1976 subRegSize(otp_srdat) -> 4;
1977 subRegSize(otp_sf) -> 1;
1978 subRegSize(lde_thresh) -> 2;
1979 subRegSize(lde_cfg1) -> 1;
1980 subRegSize(lde_ppindx) -> 2;
1981 subRegSize(lde_ppamp1) -> 2;
1982 subRegSize(lde_rxantd) -> 2;
1983 subRegSize(lde_cfg2) -> 2;
1984 subRegSize(lde_repc) -> 2;
1985 subRegSize(evc_ctrl) -> 4;
1986 subRegSize(diag_tmc) -> 2;
1987 subRegSize(pmsc_ctrl0) -> 4;
1988 subRegSize(pmsc_ctrl1) -> 4;
1989 subRegSize(pmsc_snozt) -> 1;
1990 subRegSize(pmsc_txfseq) -> 2;
1991 subRegSize(pmsc_ledc) -> 4;
1992 subRegSize(_) -> error({error}).
1993
1994 %--- Debug -----
1995
1996 debug_read(Reg, Value) ->
1997     io:format("[PmodUWB] read [16#~2.16.0B - ~w] --> ~s -> ~s~n",
1998         [regFile(Reg), Reg, debug_bitstring(Value), debug_bitstring_hex(Value)]
1999     ).
2000
2001 debug_write(Reg, Value) ->
2002     io:format("[PmodUWB] write [16#~2.16.0B - ~w] --> ~s -> ~s~n",
2003         [regFile(Reg), Reg, debug_bitstring(Value), debug_bitstring_hex(Value)]
2004     ).
2005 debug_write(Reg, SubReg, Value) ->
2006     io:format("[PmodUWB] write [16#~2.16.0B - ~w - 16#~2.16.0B - ~w] --> ~s -> ~s~
2007         n",
2008         [regFile(Reg), Reg, subReg(SubReg), SubReg, debug_bitstring(Value),
2009             debug_bitstring_hex(Value)]
2010     ).
2011
2012 debug_bitstring(Bitstring) ->
2013     lists:flatten([io_lib:format("2#~8.2.0B ", [X]) || <<X>> <= Bitstring]).
2014
2015 debug_bitstring_hex(Bitstring) ->
2016     lists:flatten([io_lib:format("16#~2.16.0B ", [X]) || <<X>> <= Bitstring]).

```

```

1 -module(pmod_uwb).
2 -behaviour(gen_server).
3
4 % API
5 -export([start_link/2]).

```

```

6 -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]).
7 -export([reception_async/0]).
8 -export([set_frame_timeout/1]).
9 -export([set_preamble_timeout/1, disable_preamble_timeout/0]).
10 -export([softreset/0, clear_rx_flags/0]).
11 -export([disable_rx/0]).
12 -export([suspend_frame_filtering/0, resume_frame_filtering/0]).
13 -export([signal_power/0]).
14 -export([prf_value/0]).
15 -export([rx_preamble_repetition/0]).
16 -export([rx_data_rate/0]).
17 -export([rx_ranging_info/0]).
18 -export([std_noise/0]).
19 -export([first_path_power_level/0]).
20 -export([get_conf/0]).
21 -export([get_rx_metadata/0]).
22
23 % gen_server callback
24 -export([init/1, handle_call/3, handle_cast/2]).
25
26 -compile({nowarn_unused_function, [debug_read/2, debug_write/2, debug_write/3,
   debug_bitstring/1, debug_bitstring_hex/1]}).
27
28 % Includes
29 -include("grisp.hrl").
30
31 -include("pmod_uwb.hrl").
32
33 %--- Macros -----
34
35 % Define the polarity and the phase of the clock
36 -define(SPI_MODE, #{clock => {low, leading}}).
37
38 -define(WRITE_ONLY_REG_FILE(RegFileID), RegFileID == tx_buffer).
39
40 -define(READ_ONLY_REG_FILE(RegFileID), RegFileID==dev_id;
41         RegFileID==sys_time;
42         RegFileID==rx_finfo;
43         RegFileID==rx_buffer;
44         RegFileID==rx_fqual;
45         RegFileID==rx_ttcko;
46         RegFileID==rx_time;
47         RegFileID==tx_time;
48         RegFileID==sys_state;
49         RegFileID==acc_mem).
50
51 %% The configurations of the subregisters of these register files are different
52 %% (some sub-registers are RO, some are RW and some have reserved bytes
53 %% that can't be written)
54 %% Thus, some registers files require to write their sub-register independently
55 %% => Write the sub-registers one by one instead of writing
56 %% the whole register file directly
57 -define(IS_SRW(RegFileID), RegFileID==agc_ctrl;
58         RegFileID==ext_sync;
59         RegFileID==ec_ctrl;
60         RegFileID==gpio_ctrl;
61         RegFileID==drx_conf;
62         RegFileID==rf_conf;
63         RegFileID==tx_cal;
64         RegFileID==fs_ctrl;
65         RegFileID==aon;

```



```

66         RegFileID==otp_if;
67         RegFileID==lde_if;
68         RegFileID==dig_diag;
69         RegFileID==pmsc).
70
71 -define(READ_ONLY_SUB_REG(SubRegister), SubRegister==irqs;
72         SubRegister==agc_stat1;
73         SubRegister==ec_rxtc;
74         SubRegister==ec_glop;
75         SubRegister==drx_car_int;
76         SubRegister==rf_status;
77         SubRegister==tc_sarl;
78         SubRegister==sarw;
79         SubRegister==tc_pg_status;
80         SubRegister==lde_thresh;
81         SubRegister==lde_ppindx;
82         SubRegister==lde_ppampl;
83         SubRegister==evc_phe;
84         SubRegister==evc_rse;
85         SubRegister==evc_fcg;
86         SubRegister==evc_fce;
87         SubRegister==evc_ffr;
88         SubRegister==evc_ovr;
89         SubRegister==evc_sto;
90         SubRegister==evc_pto;
91         SubRegister==evc_fwto;
92         SubRegister==evc_txfs;
93         SubRegister==evc_hpw;
94         SubRegister==evc_tpw).
95
96
97 %--- Types -----
98 -export_type([register_values/0]).
99
100 -type regFileID() :: atom().
101 -opaque register_values() :: map().
102
103 %--- API -----
104
105 start_link(Connector, _Opts) ->
106     gen_server:start_link({local, ?MODULE}, ?MODULE, Connector, []).
107
108
109 %% @doc read a register file
110 %%
111 %% === Example ===
112 %% To read the register file DEV_ID
113 %% '''
114 %% 1> pmod_uwb:read(dev_id).
115 %% #{model => 1, rev => 0, ridtag => "DECA", ver => 3}
116 %% '''
117 -spec read(RegFileID) -> Result when
118     RegFileID :: regFileID(),
119     Result    :: map() | {error, any()}.
120 read(RegFileID) when ?WRITE_ONLY_REG_FILE(RegFileID) ->
121     error({read_on_write_only_register, RegFileID});
122 read(RegFileID) -> call({read, RegFileID}).
123
124 %% @doc Write values in a register
125 %%
126 %% === Examples ===
127 %% To write in a simple register file (i.e. a register without any sub-register)

```

```

128 %% '''
129 %% 1> pmod_uwb:write(eui, #{eui => <<16#AAAAAABBBBBBBBBB>>}).
130 %% ok
131 %% '''
132 %% To write in one sub-register of a register file:
133 %% '''
134 %% 2> pmod_uwb:write(panadr, #{pan_id => <<16#AAAA>>}).
135 %% ok
136 %% '''
137 %% The previous code will only change the values inside the sub-register PAN_ID
138 %%
139 %% To write in multiple sub-register of a register file in the same burst:
140 %% '''
141 %% 3> pmod_uwb:write(panadr, #{pan_id => <<16#AAAA>>,
142 %%                          short_addr => <<16#BBBB>>}).
143 %% ok
144 %% '''
145 %% Some sub-registers have their own fields. For example to set the value of
146 %% the DIS_AM field in the sub-register AGC_CTRL1 of the register file AGC_CTRL:
147 %% '''
148 %% 4> pmod_uwb:write(agc_ctrl, #{agc_ctrl1 => #{dis_am => 2#0}}).
149 %% '''
150 -spec write(RegFileID, Value) -> Result when
151   RegFileID :: regFileID(),
152   Value      :: map(),
153   Result     :: ok | {error, any()}.
154 write(RegFileID, Value) when ?READ_ONLY_REG_FILE(RegFileID) ->
155   error({write_on_read_only_register, RegFileID, Value});
156 write(RegFileID, Value) when is_map(Value) ->
157   call({write, RegFileID, Value}).
158
159 %% @doc Writes the data in the TX_BUFFER register
160 %%
161 %% Value is expected to be a <b>Binary</b>
162 %% That choice was made to make the transmission of frames easier later on
163 %%
164 %% === Examples ===
165 %% Send "Hello" in the buffer
166 %% '''
167 %% 1> pmod_uwb:write_tx_data(<<"Hello">>).
168 %% '''
169 -spec write_tx_data(Value) -> Result when
170   Value      :: binary(),
171   Result     :: ok | {error, any()}.
172 write_tx_data(Value) -> call({write_tx, Value}).
173
174 %% @doc Retrieves the data received on the UWB antenna
175 %% @returns {DataLength, Data}
176 -spec get_received_data() -> Result when
177   Result     :: {integer(), bitstring()} | {error, any()}.
178 get_received_data() -> call({get_rx_data}).
179
180 get_rx_metadata() ->
181   #{rng := Rng} = read(rx_finfo),
182   #{rx_stamp := RxStamp} = read(rx_time),
183   #{tx_stamp := TxStamp} = read(tx_time),
184   #{rxtofs := Rxtofs} = read(rx_ttcko),
185   #{rxttcki := Rxttcki} = read(rx_ttkci),
186   #{snr => snr()},
187   prf => prf_value(),
188   pre => rx_preamble_repetition(),
189   data_rate => rx_data_rate(),

```

```

190     rng => Rng,
191     rx_stamp => RxStamp,
192     tx_stamp => TxStamp,
193     rxtofs => Rxtofs,
194     rxttcki => Rxttcki}.
195
196 % Source: https://forum.qorvo.com/t/how-to-calculate-the-signal-to-noise-ratio-snr-of-dw1000/5585/3
197 snr() ->
198     Delta = 87-7.5,
199     RSL = pmod_uwb:signal_power(),
200     RSL + Delta.
201
202 %% @doc Transmit data with the default options (i.e. don't wait for resp, ...)
203 %%
204 %% === Examples ===
205 %% To transmit a frame:
206 %% '''
207 %% 1> pmod_uwb:transmit(<Version:4, NextHop:8>>).
208 %% ok.
209 %% '''
210 -spec transmit(Data) -> Result when
211     Data :: bitstring(),
212     Result :: ok.
213 transmit(Data) when is_bitstring(Data) ->
214     call({transmit, Data, #tx_opts{}}),
215     wait_for_transmission().
216
217 %% @doc Performs a transmission with the specified options
218 %%
219 %% === Options ===
220 %% * wait4resp: It specifies that the reception must be enabled after
221 %%               the transmission in the expectation of a response
222 %% * w4r-tim: Specifies the turn around time in microseconds. That is the time
223 %%               the pmod will wait before enabling rx after a tx.
224 %%               Note that it won't be set if wit4resp is disabled
225 %% * txdllys: Specifies if the transmitter delayed sending should be set
226 %% * tx_delay: Specifies the delay of the transmission (see register DX_TIME)
227 %%
228 %% === Examples ===
229 %% To transmit a frame with default options:
230 %% '''
231 %% 1> pmod_uwb:transmit(<Version:4, NextHop:8>>, #tx_opts{}).
232 %% ok.
233 %% '''
234 -spec transmit(Data, Options) -> Result when
235     Data :: bitstring(),
236     Options :: tx_opts(),
237     Result :: ok.
238 transmit(Data, Options) ->
239     case Options#tx_opts.wait4resp of
240     ?ENABLED -> clear_rx_flags();
241     _ -> ok
242     end,
243     call({transmit, Data, Options}),
244     case read(sys_status) of
245     #{hdpwarn := 2#1} -> error({hdpwarn});
246     _ -> ok
247     end,
248     wait_for_transmission().
249
250 %% Wait for the transmission to be performed

```

```

251 %% usefull in the case of a delayed transmission
252 wait_for_transmission() ->
253     case read(sys_status) of
254         #{txfrs := 1} -> ok;
255         _ -> wait_for_transmission()
256     end.
257
258 %% @doc Receive data using the pmod
259 %% @equiv reception(false)
260 -spec reception() -> Result when
261     Result :: {integer(), bitstring()} | {error, any()}.
262 reception() ->
263     reception(false).
264
265 %% @doc Receive data using the pmod
266 %%
267 %% The function will hang until a frame is received on the board
268 %%
269 %% The CRC of the received frame <b>isn't</b> included in the returned value
270 %%
271 %% @param RXEnabled: specifies if the reception is already enabled on the board
272 %%                    (or set with delay)
273 %%
274 %% === Example ===
275 %% '''
276 %% 1> pmod_uwb:reception().
277 %% % Some frame is transmitted
278 %% {11, <<"Hello world">>}.
279 %% '''
280 -spec reception(RXEnabled) -> Result when
281     RXEnabled :: boolean(),
282     Result     :: {integer(), bitstring()} | {error, any()}.
283 reception(RXEnabled) ->
284     if not RXEnabled -> enable_rx();
285         true -> ok
286     end,
287     case wait_for_reception() of
288         ok ->
289             get_received_data();
290         Err ->
291             {error, Err}
292     end.
293
294 -spec reception_async() -> Result when
295     Result     :: ok | {error, any()}.
296 reception_async() ->
297     case reception() of
298         {error, _} = Err -> Err;
299         Frame ->
300             Metadata = get_rx_metadata(),
301             ieee802154_events:rx_event(Frame, Metadata)
302     end.
303
304 %% @private
305 enable_rx() ->
306     % io:format("Enabling reception~n"),
307     clear_rx_flags(),
308     call({write, sys_ctrl, #{rxenab => 2#1}}).
309
310 %% @doc Disables the reception on the pmod
311 disable_rx() ->
312     call({write, sys_ctrl, #{trxoff => 2#1}}).

```

```

313
314 wait_for_reception() ->
315     % io:format("Wait for resp-n"),
316     case read(sys_status) of
317         #{rxrfto := 1} -> rxrfto;
318         #{rxphe := 1} -> rxphe;
319         #{rxfce := 1} -> rxfce;
320         #{rxrfs1 := 1} -> rxrfs1;
321         #{rxpto := 1} -> rxpto;
322         #{rxsfdto := 1} -> rxsfdto;
323         #{ldeerr := 1} -> ldeerr;
324         #{affrej := 1} -> affrej;
325         #{rxdfdr := 0} -> wait_for_reception();
326         #{rxfce := 1} -> rxfce;
327         #{rxfcg := 1} -> ok;
328         #{rxfcg := 0} -> wait_for_reception();
329         % #{rxdfdr := 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
330     _ -> error({error_wait_for_reception})
331     end.
332
333 %% @doc Set the frame wait timeout and enables it
334 %% The unit is roughly 1us (cf. user manual)
335 %% If a float is given, it's decimal part is removed using trunc/1
336 %% @end
337 -spec set_frame_timeout(Timeout) -> Result when
338     Timeout :: microseconds(),
339     Result  :: ok.
340 set_frame_timeout(Timeout) when is_float(Timeout) ->
341     set_frame_timeout(trunc(Timeout));
342 set_frame_timeout(Timeout) when is_integer(Timeout) ->
343     write(rx_fwto, #{rxfwto => Timeout}),
344     write(sys_cfg, #{rxwtoc => 2#1}). % enable receive wait timeout
345
346 %% @doc Sets the preamble timeout. (PRETOC register of the DW1000)
347 %% The unit of 'Timeout' is in units usec
348 %% If the value is a float, trunc is called to remove the decimal part
349 %% Internally, it's converted in units of PAC size
350 -spec set_preamble_timeout(Timeout) -> ok when
351     Timeout :: non_neg_integer().
352 set_preamble_timeout(T0) when is_float(T0) ->
353     set_preamble_timeout(trunc(T0));
354 set_preamble_timeout(T0) when is_integer(T0) ->
355     call({preamble_timeout, T0}),
356     write(drx_conf, #{drx_pretoc => 0}).
357
358 disable_preamble_timeout() ->
359     write(drx_conf, #{drx_pretoc => 0}).
360
361 %% @doc Performs a reset of the IC following the procedure (cf. sec. 7.2.50.1)
362 softreset() ->
363     write(pmsc, #{pmsc_ctrl0 => #{sysclks => 2#01}}),
364     write(pmsc, #{pmsc_ctrl0 => #{softrest => 16#0}}),
365     write(pmsc, #{pmsc_ctrl0 => #{softreset => 16#FFFF}}).
366
367
368 clear_rx_flags() ->
369     write(sys_status, #{rxsfdto => 2#1,
370         rxpto => 2#1,
371         rxrfto => 2#1,
372         rxrfs1 => 2#1,

```

```

373         rxfce => 2#1,
374         rxphe => 2#1,
375         rxprd => 2#1,
376         rxdsfdd => 2#1,
377         rxphd => 2#1,
378         rxdfc => 2#1,
379         rxfcg => 2#1}).
380
381 suspend_frame_filtering() ->
382     write(sys_cfg, #{ffcn => 2#0}).
383
384 resume_frame_filtering() ->
385     write(sys_cfg, #{ffcn => 2#1}).
386
387 %% @doc Returns the estimated value of the signal power in dBm
388 %% cf. user manual section 4.7.2
389 signal_power() ->
390     C = channel_impulse_resp_pow() , % Channel impulse response power value (
391         CIR_PWR)
392     A = case prf_value() of
393         16 -> 113.77;
394         64 -> 121.74
395     end, % Constant. For PRF of 16 MHz = 113.77, for PRF of 64MHz = 121.74
396     N = preamble_acc(), % Preamble accumulation count value (RXPACC but might be
397         adjusted)
398     % io:format("C: ~w-n A:~w-n N:~w-n", [C, A, N]),
399     Res = 10 * math:log10((C * math:pow(2, 17))/math:pow(N, 2)) - A,
400     % io:format("Estimated signal power: ~p dBm~n", [Res]),
401     % io:format("Std noise: ~w-n", [pmod_uwb:read(rx_fqual)]),
402     Res.
403
404 preamble_acc() ->
405     #{rxpacc := RXPACC} = read(rx_finfo),
406     #{rxpacc_nosat := RXPACC_NOSAT} = read(drx_conf),
407     if
408         RXPACC == RXPACC_NOSAT -> RXPACC - 5;
409         true -> RXPACC
410     end.
411
412 channel_impulse_resp_pow() ->
413     #{cir_pwr := CIR_PWR} = read(rx_fqual),
414     CIR_PWR.
415
416 %% @doc Gives the value of the PRF in MHz
417 -spec prf_value() -> 16 | 64.
418 prf_value() ->
419     #{agc_tune1 := AGC_TUNE1} = read(agc_ctrl),
420     case AGC_TUNE1 of
421         16#8870 -> 16;
422         16#889B -> 64
423     end.
424
425 %% @doc returns the preamble symbols repetition
426 rx_preamble_repetition() ->
427     #{rxpsr := RXPSR} = read(rx_finfo),
428     case RXPSR of
429         0 -> 16;
430         1 -> 64;
431         2 -> 1024;
432         3 -> 4096
433     end.

```

```

433 %% @doc returns the data rate of the received frame in kbps
434 rx_data_rate() ->
435     #{rxbr := RXBR} = read(rx_finfo),
436     case RXBR of
437         0 -> 110;
438         1 -> 850;
439         2 -> 6800
440     end.
441
442 % @doc returns the value of the 'Ranging' bit of the received frame
443 rx_ranging_info() ->
444     #{rng := RNG} = read(rx_finfo),
445     RNG.
446
447 std_noise() ->
448     #{std_noise := STD_NOISE} = read(rx_fqual),
449     STD_NOISE.
450
451 first_path_power_level() ->
452     #{fp_ampl1 := F1} = read(rx_time),
453     #{fp_ampl2 := F2, pp_ampl3 := F3} = read(rx_fqual),
454     A = 113.77,
455     N = preamble_acc(),
456     10 * math:log10((math:pow(F1,2) + math:pow(F2, 2) + math:pow(F3, 2))/math:pow(
457         N, 2)) - A.
458
459 get_conf() ->
460     call({get_conf}).
461
462 %--- gen_server Callbacks -----
463 %% @private
464 init(Slot) ->
465     % Verify the slot used
466     case {grisp_hw:platform(), Slot} of
467         {grisp2, spi2} -> ok;
468         {P, S} -> error({incompatible_slot, P, S})
469     end,
470     grisp_devices:register(Slot, ?MODULE),
471     Bus = grisp_spi:open(Slot),
472     case verify_id(Bus) of
473         ok -> softreset(Bus);
474         Val -> error({dev_id_no_match, Val})
475     end,
476     ldeload(Bus),
477     % TODO Merge the next 4 cfg commands into one
478     write_default_values(Bus),
479     config(Bus),
480     setup_sfd(Bus),
481     Conf = #phy_cfg{},
482     {ok, #{bus => Bus, conf => Conf}}.
483
484 %% @private
485 handle_call({read, RegFileID}, _From, #{bus := Bus} = State) ->
486     {reply, read_reg(Bus, RegFileID), State};
487 handle_call({write, RegFileID, Value}, _From, #{bus := Bus} = State) ->
488     {reply, write_reg(Bus, RegFileID, Value), State};
489 handle_call({write_tx, Value}, _From, #{bus := Bus} = State) ->
490     {reply, write_tx_data(Bus, Value), State};
491 handle_call({transmit, Data, Options}, _From, #{bus := Bus} = State) ->
492     {reply, tx(Bus, Data, Options), State};
493 handle_call({delayed_transmit, Data, Delay}, _From, #{bus := Bus} = State) ->

```

```

494     {reply, delayed_tx(Bus, Data, Delay), State};
495 handle_call({get_rx_data}, _From, #{bus := Bus} = State) ->
496     {reply, get_rx_data(Bus), State};
497 handle_call({get_conf}, _From, #{conf := Conf} = State) ->
498     {reply, Conf, State};
499 handle_call({preamble_timeout, T0us}, _From, State) ->
500     #{bus := Bus, conf := Conf} = State,
501     PACSize = Conf#phy_cfg.pac_size,
502     case T0us of
503     0 ->
504         write_reg(Bus, drx_conf, #{drx_pretoc => 0});
505     _ ->
506         % Remove 1 because DW1000 counter auto. adds 1 (cf. 7.2.40.9 user
507         % manual)
508         To = math:ceil(T0us / PACSize)-1,
509         write_reg(Bus, drx_conf, #{drx_pretoc => round(To)})
510     end,
511     {reply, ok, State};
512 handle_call(Request, _From, _State) ->
513     error({unknown_call, Request}).
514 %% @private
515 handle_cast(Request, _State) -> error({unknown_cast, Request}).
516
517 %--- Internal -----
518
519 call(Call) ->
520     Dev = grisp_devices:default(?MODULE),
521     gen_server:call(Dev#device.pid, Call).
522
523
524 %% @doc Verify the dev_id register of the pmod
525 %% @returns ok if the value is correct, otherwise the value read
526 verify_id(Bus) ->
527     #{ridtag := RIDTAG, model := MODEL} = read_reg(Bus, dev_id),
528     case {RIDTAG, MODEL} of
529     {"DECA", 1} -> ok;
530     _ -> {RIDTAG, MODEL}
531     end.
532
533 %% @private
534 %% Performs a softreset on the pmod
535 -spec softreset(Bus::grisp_spi:ref()) -> ok.
536 softreset(Bus) ->
537     write_reg(Bus, pmsc, #{pmsc_ctrl0 => #{sysclks => 2#01}}),
538     write_reg(Bus, pmsc, #{pmsc_ctrl0 => #{softrest => 16#0}}),
539     write_reg(Bus, pmsc, #{pmsc_ctrl0 => #{softreset => 16#FFFF}}).
540
541 %% @private
542 %% Writes the default values described in section 2.5.5 of the user manual
543 -spec write_default_values(Bus::grisp_spi:ref()) -> ok.
544 write_default_values(Bus) ->
545     write_reg(Bus, lde_if, #{lde_cfg1 => #{ntm => 16#D}, lde_cfg2 => 16#1607}),
546     write_reg(Bus, agc_ctrl, #{agc_tune1 => 16#8870, agc_tune2 => 16#2502A907}),
547     write_reg(Bus, drx_conf, #{drx_tune2 => 16#311A002D}),
548     write_reg(Bus, tx_power, #{tx_power => 16#0E082848}),
549     write_reg(Bus, rf_conf, #{rf_txctrl => 16#001E3FE3}),
550     write_reg(Bus, tx_cal, #{tc_pgdelay => 16#B5}),
551     write_reg(Bus, fs_ctrl, #{fs_plltune => 16#BE}).
552
553 %% @private
554 config(Bus) ->

```



```

555     write_reg(Bus, ext_sync, #{ec_ctrl => #{pll1dt => 2#1}},
556     %write_reg(Bus, pmisc, #{pmisc_ctrl1 => #{lderune => 2#0}},
557     % Now enable RX and TX leds
558     write_reg(Bus, gpio_ctrl, #{gpio_mode => #{msgp2 => 2#01, msgp3 => 2#01}},
559     % Enable RXOK and SFD leds
560     write_reg(Bus, gpio_ctrl, #{gpio_mode => #{msgp0 => 2#01, msgp1 => 2#01}},
561     write_reg(Bus, pmisc, #{pmisc_ctrl0 => #{gpdce => 2#1, khzclken => 2#1}},
562     write_reg(Bus, pmisc, #{pmisc_ledc => #{blnken => 2#1}},
563     write_reg(Bus, dig_diag, #{evc_ctrl => #{evc_en => 2#1}}, % enable counting
        event for debug purposes
564     % write_reg(Bus, sys_cfg, #{rxwtoe => 2#1}),
565     write_reg(Bus, tx_fctrl, #{txpsr => 2#10}). % Setting preamble symbols to 1024
566
567 %% @private
568 %% Load the microcode from ROM to RAM
569 %% It follows the steps described in section 2.5.5.10 of the DW1000 user manual
570 ldeload(Bus) ->
571     write_reg(Bus, pmisc, #{pmisc_ctrl0 => #{sysclks => 2#01}},
572     write_reg(Bus, pmisc, #{pmisc_ctrl0 => #{otp => 2#1, res8 => 2#1}}, % Writes 0
        x0301 in pmisc_ctrl0
573     write_reg(Bus, otp_if, #{otp_ctrl => #{ldeload => 2#1}}, % Writes 0x8000 in
        OTP_CTRL
574     timer:sleep(150), % User manual requires a wait of 150 s
575     write_reg(Bus, pmisc, #{pmisc_ctrl0 => #{sysclks => 2#0}}, % Writes 0x0200 in
        pmisc_ctrl0
576     write_reg(Bus, pmisc, #{pmisc_ctrl0 => #{res8 => 2#0}}).
577
578 %% @private
579 %% If no frame is transmitted before AUTOACK, then the SFD isn't properly set
580 %% (cf. section 5.3.1.2 SFD initialisation)
581 setup_sfd(Bus) ->
582     write_reg(Bus, sys_ctrl, #{txsstr => 2#1, trxoff => 2#1}).
583
584 %% @private
585 %% Transmit the data using UWB
586 %% @param Options is used to set options about the transmission like a
        transmission delay, etc.
587 -spec tx(grisp_spi:ref(), Data :: binary(), Options :: #tx_opts{}) -> ok.
588 tx(Bus, Data, #tx_opts{wait4resp = Wait4resp, w4r_tim = W4rTim, txdlys = TxDlys,
        tx_delay = TxDelay, ranging = Ranging}) ->
589     % Writing the data that will be sent (w/o CRC)
590     DataLength = byte_size(Data) + 2, % DW1000 automatically adds the 2 bytes CRC
591     write_tx_data(Bus, Data),
592     % Setting the options of the transmission
593     case Wait4resp of
594         ?ENABLED -> write_reg(Bus, ack_resp_t, #{w4r_tim => W4rTim});
595         _ -> ok
596     end,
597     case TxDlys of
598         ?ENABLED -> write_reg(Bus, dx_time, #{dx_time => TxDelay});
599         _ -> ok
600     end,
601     write_reg(Bus, tx_fctrl, #{txboffs => 2#0, tr => Ranging, tflen => DataLength}
        ),
602     write_reg(Bus, sys_ctrl, #{txsstr => 2#1, wait4resp => Wait4resp, txdlys =>
        TxDlys}). % start transmission and some options
603
604 %% @private
605 %% Transmit the data with a specified delay using UWB
606 delayed_tx(Bus, Data, Delay) ->
607     write_reg(Bus, dx_time, #{dx_time => Delay}),
608     DataLength = byte_size(Data) + 2, % DW1000 automatically adds the 2 bytes CRC

```

```

609     write_tx_data(Bus, Data),
610     write_reg(Bus, tx_fctrl, #{txboffs => 2#0, tflen => DataLength}),
611     write_reg(Bus, sys_ctrl, #{txstrt => 2#1, txdlys => 2#1}). % start
        transmission
612
613 %% @private
614 %% Get the received data (without the CRC bytes) stored in the rx_buffer
615 get_rx_data(Bus) ->
616     #{rxflen := FrameLength} = read_reg(Bus, rx_finfo),
617     Frame = read_rx_data(Bus, FrameLength-2), % Remove the CRC bytes
618     {FrameLength, Frame}.
619
620 %% @private
621 %% @doc Reverse the byte order of the bitstring given in the argument
622 %% @param Bin a bitstring
623 reverse(Bin) -> reverse(Bin, <<>>).
624 reverse(<<Bin:8>>, Acc) ->
625     <<Bin, Acc/binary>>;
626 reverse(<<Bin:8, Rest/bitstring>>, Acc) ->
627     reverse(Rest, <<Bin, Acc/binary>>).
628
629 % Source: https://stackoverflow.com/a/43310493
630 % reverse(Binary) ->
631 %     Size = bit_size(Binary),
632 %     <<X:Size/integer-little>> = Binary,
633 %     <<X:Size/integer-big>>.
634
635 %% @private
636 %% @doc Creates the header of the SPI transaction between the GRiSP and the pmod
637 %%
638 %% It creates a header of 1 bytes. The header is used in a transaction that will
        affect
639 %% the whole register file (read/write)
640 %%
641 %% @param Op an atom (either <i>read</i> or <i>write</i>)
642 %% @param RegFileID an atom representing the register file
643 %% @returns a formatted header of <b>1 byte</b> long as described in the user
        manual
644 header(Op, RegFileID) ->
645     <<(rw(Op)):1, 2#0:1, (regFile(RegFileID)):6>>.
646
647 %% @private
648 %% @doc Creates the header of the SPI transaction between the GRiSP and the pmod
649 %%
650 %% It creates a header of 2 bytes. The header is used in a transaction that will
        affect
651 %% the whole sub-register (read/write)
652 %% Careful: The sub-register needs to be mapped in the hrl file
653 %%
654 %% @param Op an atom (either <i>read</i> or <i>write</i>)
655 %% @param RegFileID an atom representing the register file
656 %% @param SubRegister an atom representing the sub-register
657 %% @returns a formatted header of <b>2 byte</b> long as described in the user
        manual
658 header(Op, RegFileID, SubRegister) ->
659     case subReg(SubRegister) < 127 of
660         true -> header(Op, RegFileID, SubRegister, 2);
661         _ -> header(Op, RegFileID, SubRegister, 3)
662     end.
663
664 header(Op, RegFileID, SubRegister, 2) ->
665     <<(rw(Op)):1, 2#1:1, (regFile(RegFileID)):6,

```

```

666     2#0:1, (subReg(SubRegister)):7 >>;
667 header(Op, RegFileID, SubRegister, 3) ->
668     <<_:1, HighOrder:8, LowOrder:7>> = <<(subReg(SubRegister)):16>>,
669     << (rw(Op)):1, 2#1:1, (regFile(RegFileID)):6,
670     2#1:1, LowOrder:7,
671     HighOrder:8>>.
672
673 %% @private
674 %% @doc Read the values stored in a register file
675 read_reg(Bus, lde_ctrl) -> read_reg(Bus, lde_if);
676 read_reg(Bus, lde_if) ->
677     lists:foldl(fun(Elem, Acc) ->
678         Res = read_sub_reg(Bus, lde_if, Elem),
679         maps:merge(Acc, Res)
680     end,
681     #{}),
682     [lde_thresh, lde_cfg1, lde_ppindx, lde_ppamp1, lde_rxantd,
        lde_cfg2, lde_repc]);
683 read_reg(Bus, RegFileID) ->
684     Header = header(read, RegFileID),
685     [Resp] = grisp_spi:transfer(Bus, [{?SPI_MODE, Header, 1, regSize(RegFileID)}])
686     ,
687     % debug_read(RegFileID, Resp),
688     reg(decode, RegFileID, Resp).
689
690 read_sub_reg(Bus, RegFileID, SubRegister) ->
691     Header = header(read, RegFileID, SubRegister),
692     HeaderSize = byte_size(Header),
693     % io:format("[HEADER] type ~w - ~w - ~w~n", [HeaderSize, Header, subRegSize(
        SubRegister)]),
694     [Resp] = grisp_spi:transfer(Bus, [{?SPI_MODE, Header, HeaderSize, subRegSize(
        SubRegister)}]),
695     reg(decode, SubRegister, Resp).
696
697
698 %% @doc get the received data
699 %% @param Length is the total length of the data we are trying to read
700 read_rx_data(Bus, Length) ->
701     Header = header(read, rx_buffer),
702     [Resp] = grisp_spi:transfer(Bus, [{?SPI_MODE, Header, 1, Length}]),
703     Resp.
704
705 % TODO: check that user isn't trying to write reserved bits by passing res, res1,
706     ... in the map fields
707 %% @doc used to write the values in the map given in the Value argument
708 -spec write_reg(Bus::grisp_spi:ref(), RegFileID::regFileID(), Value::map()) -> ok.
709 % Write each sub-register one by one.
710 % If the user tries to write in a read-only sub-register, an error is thrown
711 write_reg(Bus, RegFileID, Value) when ?IS_SRW(RegFileID) ->
712     maps:map(
713     fun(SubRegister, Val) ->
714         CurrVal = maps:get(SubRegister, read_reg(Bus, RegFileID)), % ? can the
715         read be done before ? Maybe but not assured that no values
716         changes after a write in the register
717         Body = case CurrVal of
718             V when is_map(V) -> reg(encode, SubRegister, maps:
719                 merge_with(fun(_Key, _Old, New) -> New end, CurrVal,
720                 Val));
721             _ -> reg(encode, SubRegister, #{SubRegister => Val})
722         end,
723     Header = header(write, RegFileID, SubRegister),

```

```

719         % debug_write(RegFileID, SubRegister, Body),
720         _ = grisp_spi:transfer(Bus, [{?SPI_MODE, <<Header/binary, Body/binary
721             >>, 2+subRegSize(SubRegister), 0}])
722     end,
723     Value),
724     ok;
725 write_reg(Bus, RegFileID, Value) ->
726     Header = header(write, RegFileID),
727     CurrVal = read_reg(Bus, RegFileID),
728     ValuesToWrite = maps:merge_with(fun(_Key, _Value1, Value2) -> Value2 end,
729         CurrVal, Value),
730     Body = reg(encode, RegFileID, ValuesToWrite),
731     % debug_write(RegFileID, Body),
732     _ = grisp_spi:transfer(Bus, [{?SPI_MODE, <<Header/binary, Body/binary>>, 1+
733         regSize(RegFileID), 0}]),
734     ok.
735 %% @doc write_tx_data/2 sends data (Value) in the register tx_buffer
736 %% @param Value is the data to be written. It must be a binary and have a size of
737 %% maximum 1024 bits
738 write_tx_data(Bus, Value) when is_binary(Value), (bit_size(Value) < 1025) ->
739     Header = header(write, tx_buffer),
740     Length = byte_size(Value),
741     % debug_write(tx_buffer, Body),
742     _ = grisp_spi:transfer(Bus, [{?SPI_MODE, <<Header/binary, Value/binary>>, 1+
743         Length, 0}]),
744     ok.
745 %---- Register mapping -----
746 %% @doc Used to either decode the data returned by the pmod or to encode to data
747 %% that will be sent to the pmod
748 %%
749 %% The transmission on the MISO line is done byte by byte starting from the lowest
750 %% rank byte to the highest rank
751 %% Example: dev_id value is 0xDECA0130 but 0x3001CADE is transmitted over the MISO
752 %% line
753 -spec reg(Type, Register, Val) -> Ret when
754     Type      :: encode | decode,
755     Register  :: regFileID(),
756     Val       :: nonempty_binary() | register_values(),
757     Ret       :: nonempty_binary() | register_values().
758 reg(encode, SubRegister, Value) when ?READ_ONLY_SUB_REG(SubRegister) -> error({
759     writing_read_only_sub_register, SubRegister, Value});
760 reg(decode, dev_id, Resp) ->
761     <<
762     RIDTAG:16, Model:8, Ver:4, Rev:4
763     >> = reverse(Resp),
764     #{
765         ridtag => integer_to_list(RIDTAG, 16), model => Model, ver => Ver, rev =>
766         Rev
767     };
768 reg(decode, eui, Resp) ->
769     #{
770         eui => reverse(Resp)
771     };
772 reg(encode, eui, Val) ->
773     #{
774         eui := EUI
775     } = Val,
776     reverse(
777         EUI

```

```

771 );
772 reg(decode, panadr, Resp) ->
773 <<
774     PanId:16, ShortAddr:16
775 >> = reverse(Resp),
776 #{
777     pan_id => <<PanId:16>>, short_addr => <<ShortAddr:16>>
778 };
779 reg(encode, panadr, Val) ->
780 #{
781     pan_id := PanId, short_addr := ShortAddr
782 } = Val,
783 reverse(<<
784     PanId:16/bitstring, ShortAddr:16/bitstring
785 >>);
786 reg(decode, sys_cfg, Resp) ->
787 <<
788     FFA4:1, FFAR:1, FFAM:1, FFAA:1, FFAD:1, FFAB:1, FFBC:1, FFEN:1, % bits 7-0
789     FCS_INIT2F:1, DIS_RSDE:1, DIS_PHE:1, DIS_DRXB:1, DIS_FCE:1, SPI_EDGE:1,
790     HIRQ_POL:1, FFA5:1, % bits 15-8
791     _:1, RXM110K:1, _:3, DIS_STXP:1, PHR_MODE:2, % bits 23-16
792     AACKPEND:1, AUTOACK:1, RXAUTR:1, RXWTOE:1, _:4 % bits 31-24
793 >> = Resp,
794 #{
795     aackpend => AACKPEND, autoack => AUTOACK, rxautr => RXAUTR, rxwtoe =>
796     RXWTOE,
797     rxm110k => RXM110K, dis_stxp => DIS_STXP, phr_mode => PHR_MODE,
798     fcs_init2f => FCS_INIT2F, dis_rsde => DIS_RSDE, dis_phe => DIS_PHE,
799     dis_drxb => DIS_DRXB, dis_fce => DIS_FCE, spi_edge => SPI_EDGE,
800     hirq_pol => HIRQ_POL, ffa5 => FFA5,
801     ffa4 => FFA4, ffar => FFAR, ffam => FFAM, ffaa => FFAA, ffad => FFAD, ffab
802     => FFAB, ffbc => FFBC, ffen => FFEN
803 };
804 reg(encode, sys_cfg, Val) ->
805 #{
806     aackpend := AACKPEND, autoack := AUTOACK, rxautr := RXAUTR, rxwtoe :=
807     RXWTOE,
808     rxm110k := RXM110K, dis_stxp := DIS_STXP, phr_mode := PHR_MODE,
809     fcs_init2f := FCS_INIT2F, dis_rsde := DIS_RSDE, dis_phe := DIS_PHE,
810     dis_drxb := DIS_DRXB, dis_fce := DIS_FCE, spi_edge := SPI_EDGE,
811     hirq_pol := HIRQ_POL, ffa5 := FFA5,
812     ffa4 := FFA4, ffar := FFAR, ffam := FFAM, ffaa := FFAA, ffad := FFAD, ffab
813     := FFAB, ffbc := FFBC, ffen := FFEN
814 } = Val,
815 <<
816     FFA4:1, FFAR:1, FFAM:1, FFAA:1, FFAD:1, FFAB:1, FFBC:1, FFEN:1, % bits 7-0
817     FCS_INIT2F:1, DIS_RSDE:1, DIS_PHE:1, DIS_DRXB:1, DIS_FCE:1, SPI_EDGE:1,
818     HIRQ_POL:1, FFA5:1, % bits 15-8
819     2#0:1, RXM110K:1, 2#0:3, DIS_STXP:1, PHR_MODE:2, % bits 23-16
820     AACKPEND:1, AUTOACK:1, RXAUTR:1, RXWTOE:1, 2#0:4 % bits 31-24
821 >>;
822 reg(decode, sys_time, Resp) ->
823 <<
824     SysTime:40
825 >> = reverse(Resp),
826 #{
827     sys_time => SysTime
828 };
829 reg(decode, tx_fctrl, Resp) ->
830 <<
831     IFSDELAY:8, TXBOFFS:10, PE:2, TXPSR:2, TXPRF:2, TR:1, TXBR:2, R:3, TFLE:3,
832     TFLEN:7

```

```

822     >> = reverse(Resp),
823     #{
824         ifsdelay => IFSDELAY, txboffs => TXBOFFS, pe => PE, txpsr => TXPSR, txprf
            => TXPRF, tr => TR, txbr => TXBR, r => R, tfle => TFLE, tfllen => TFLEN
825     };
826 reg(encode, tx_fctrl, Val) ->
827     #{
828         ifsdelay := IFSDELAY, txboffs := TXBOFFS, pe := PE, txpsr := TXPSR, txprf
            := TXPRF, tr := TR, txbr := TXBR, r := R, tfle := TFLE, tfllen := TFLEN
829     } = Val,
830     reverse(<<
831         IFSDELAY:8, TXBOFFS:10, PE:2, TXPSR:2, TXPRF:2, TR:1, TXBR:2, R:3, TFLE:3,
            TFLEN:7
832     >>);
833 % TX_BUFFER is write only => no decode
834 reg(decode, dx_time, Resp) ->
835     #{
836         dx_time => reverse(Resp)
837     };
838 reg(encode, dx_time, Val) ->
839     #{
840         dx_time := DX_TIME
841     } = Val,
842     reverse(<<
843         DX_TIME:40
844     >>);
845 reg(decode, rx_fwto, Resp) ->
846     <<
847         RXFWTO:16
848     >> = reverse(Resp),
849     #{
850         rxfwto => RXFWTO
851     };
852 reg(encode, rx_fwto, Val) ->
853     #{
854         rxfwto := RXFWTO
855     } = Val,
856     reverse(<<
857         RXFWTO:16
858     >>);
859 reg(decode, sys_ctrl, Resp) ->
860     <<
861         WAIT4RESP:1, TRXOFF:1, _:2, CANSFCS:1, TXDLYS:1, TXSTRT:1, SFCST:1, % bits
            7-0
862         _:6, RXDLYE:1, RXENAB:1, % bits 15-8
863         _:8, % bits 23-16
864         _:7, HRBPT:1 % bits 31-24
865     >> = Resp,
866     #{
867         sfcst => SFCST, txstrt => TXSTRT, txdlys => TXDLYS, cansfcs => CANSFCS,
            trxoff => TRXOFF, wait4resp => WAIT4RESP,
868         rxenab => RXENAB, rxdlye => RXDLYE,
869         hrbpt => HRBPT
870     };
871 reg(encode, sys_ctrl, Val) ->
872     #{
873         sfcst := SFCST, txstrt := TXSTRT, txdlys := TXDLYS, cansfcs := CANSFCS,
            trxoff := TRXOFF, wait4resp := WAIT4RESP,
874         rxenab := RXENAB, rxdlye := RXDLYE,
875         hrbpt := HRBPT
876     } = Val,
877     <<

```

```

878     WAIT4RESP:1, TRXOFF:1, 2#0:2, CANSFCS:1, TXDLYS:1, TXSTRT:1, SFCST:1, %
      bits 7-0
879     2#0:6, RXDLYE:1, RXENAB:1, % bits 15-8
880     2#0:8, % bits 23-16
881     2#0:7, HRBPT:1 % bits 31-24
882     >>;
883 reg(decode, sys_mask, Resp) ->
884     <<
885     MTXFRS:1, MTXPHS:1, MTXPRS:1, MTXFRB:1, MAAT:1, MESYNCR:1, MCPLOCK:1,
      Reserved0:1, % bits 7-0
886     MRXFCE:1, MRXFCG:1, MRXDFR:1, MRXPHE:1, MRXPHD:1, MLDEDON:1, MRXSFD:1,
      MRXPRD:1, % bits 15-8
887     MSLP2INIT:1, MGPIOIRQ:1, MRXPTO:1, MRXOVRR:1, Reserved1:1, MLDEERR:1,
      MRXRFTO:1, MRXRFSL:1, % bits 23-16
888     Reserved2:2, MAFFREJ:1, MTXBERR:1, MHPDDWAR:1, MPLLHILO:1, MCPLLLL:1,
      MRFPLLLL:1 % bits 31-24
889     >> = Resp,
890     #{
891     mtxfrs => MTXFRS, mtjspxs => MTXPHS, mtjspxrs => MTXPRS, mtjspxrb => MTXFRB,
      maat => MAAT, mesynchr => MESYNCR, mcplck => MCPLOCK, res0 =>
      Reserved0, % bits 7-0
892     mrxfce => MRXFCE, mrxfcg => MRXFCG, mrxdf => MRXDFR, mrxphe => MRXPHE,
      mrxphd => MRXPHD, mldeon => MLDEDON, mrxsfd => MRXSFD, mrxprd =>
      MRXPRD, % bits 15-8
893     mslp2init => MSLP2INIT, mgpioirq => MGPIOIRQ, mrxpto => MRXPTO, mrxovrr =>
      MRXOVRR, res1 => Reserved1, mldeerr => MLDEERR, mrxrfto => MRXRFTO,
      mrxrfsl => MRXRFSL, % bits 23-16
894     res2 => Reserved2, maffrej => MAFFREJ, mtxberr => MTXBERR, mhpddwar =>
      MHPDDWAR, mpllhilo => MPLLHILO, mcpllll => MCPLLLL, mrfpllll =>
      MRFPLLLL % bits 31-24
895     };
896 reg(encode, sys_mask, Val) ->
897     #{
898     mtxfrs := MTXFRS, mtjspxs := MTXPHS, mtjspxrs := MTXPRS, mtjspxrb := MTXFRB,
      maat := MAAT, mesynchr := MESYNCR, mcplck := MCPLOCK, res0 :=
      Reserved0, % bits 7-0
899     mrxfce := MRXFCE, mrxfcg := MRXFCG, mrxdf := MRXDFR, mrxphe := MRXPHE,
      mrxphd := MRXPHD, mldeon := MLDEDON, mrxsfd := MRXSFD, mrxprd :=
      MRXPRD, % bits 15-8
900     mslp2init := MSLP2INIT, mgpioirq := MGPIOIRQ, mrxpto := MRXPTO, mrxovrr :=
      MRXOVRR, res1 := Reserved1, mldeerr := MLDEERR, mrxrfto := MRXRFTO,
      mrxrfsl := MRXRFSL, % bits 23-16
901     res2 := Reserved2, maffrej := MAFFREJ, mtxberr := MTXBERR, mhpddwar :=
      MHPDDWAR, mpllhilo := MPLLHILO, mcpllll := MCPLLLL, mrfpllll :=
      MRFPLLLL % bits 31-24
902     } = Val,
903     <<
904     MTXFRS:1, MTXPHS:1, MTXPRS:1, MTXFRB:1, MAAT:1, MESYNCR:1, MCPLOCK:1,
      Reserved0:1, % bits 7-0
905     MRXFCE:1, MRXFCG:1, MRXDFR:1, MRXPHE:1, MRXPHD:1, MLDEDON:1, MRXSFD:1,
      MRXPRD:1, % bits 15-8
906     MSLP2INIT:1, MGPIOIRQ:1, MRXPTO:1, MRXOVRR:1, Reserved1:1, MLDEERR:1,
      MRXRFTO:1, MRXRFSL:1, % bits 23-16
907     Reserved2:2, MAFFREJ:1, MTXBERR:1, MHPDDWAR:1, MPLLHILO:1, MCPLLLL:1,
      MRFPLLLL:1 % bits 31-24
908     >>;
909 reg(decode, sys_status, Resp) ->
910     <<
911     TXFRS:1, TXPHS:1, TXPRS:1, TXFRB:1, AAT:1, ESYNCR:1, CPLOCK:1, IRQS:1, %
      bits 7-0
912     RXFCE:1, RXFCG:1, RXDFR:1, RXPHE:1, RXPHD:1, LDEDONE:1, RXSFD:1, RXPRD:1,
      % bits 15-8

```

```

913     SPL2INIT:1, GPIOIRQ:1, RXPTO:1, RXOVR:1, Reserved0:1, LDEERR:1, RXRFTO:1,
914     RXRFSL:1, % bits 23-16
915     ICRBP:1, HSRBP:1, AFFREJ:1, TXBERR:1, HPDWARN:1, RXSFDTO:1, CLCKPLL_LL:1,
916     RFPLL_LL:1, % bits 31-24
917     Reserved1:5, TXPUTE:1, RXPRES:1, RXRSCS:1 % bits 39-32
918 >> = Resp,
919 # {
920     txfrs => TXFRS, txphs => TXPHS, txprs => TXPRS, txfrb => TXFRB, aat => AAT
921     , esyncr => ESYNCR, cplock => CPLOCK, irqs => IRQS, % bits 7-0
922     rxfce => RXFCE, rxfcg => RXFCG, rxdfrr => RXDFR, rxphe => RXPHE, rxphd =>
923     RXPHD, ldedone => LDEDONE, rxsfdd => RXSFDD, rxprd => RXPRD, % bits
924     15-8
925     spl2init => SPL2INIT, gpioirq => GPIOIRQ, rxpto => RXPTO, rxovrr =>
926     RXOVR, res0 => Reserved0, ldeerr => LDEERR, rxrfto => RXRFTO, rxrfs1
927     => RXRFSL, % bits 23-16
928     icrbp => ICRBP, hsrbp => HSRBP, affrej => AFFREJ, txberr => TXBERR,
929     hdpwarn => HPDWARN, rxsfdto => RXSFDTO, clkpll_ll => CLCKPLL_LL,
930     rfppll_ll => RFPLL_LL, % bits 31-24
931     res1 => Reserved1, txpute => TXPUTE, rxprej => RXPRES, rxrscs => RXRSCS
932 }
933 reg(encode, sys_status, Val) ->
934 # {
935     txfrs := TXFRS, txphs := TXPHS, txprs := TXPRS, txfrb := TXFRB, aat := AAT
936     , esyncr := ESYNCR, cplock := CPLOCK, irqs := IRQS, % bits 7-0
937     rxfce := RXFCE, rxfcg := RXFCG, rxdfrr := RXDFR, rxphe := RXPHE, rxphd :=
938     RXPHD, ldedone := LDEDONE, rxsfdd := RXSFDD, rxprd := RXPRD, % bits
939     15-8
940     spl2init := SPL2INIT, gpioirq := GPIOIRQ, rxpto := RXPTO, rxovrr :=
941     RXOVR, res0 := Reserved0, ldeerr := LDEERR, rxrfto := RXRFTO, rxrfs1
942     := RXRFSL, % bits 23-16
943     icrbp := ICRBP, hsrbp := HSRBP, affrej := AFFREJ, txberr := TXBERR,
944     hdpwarn := HPDWARN, rxsfdto := RXSFDTO, clkpll_ll := CLCKPLL_LL,
945     rfppll_ll := RFPLL_LL, % bits 31-24
946     res1 := Reserved1, txpute := TXPUTE, rxprej := RXPRES, rxrscs := RXRSCS
947 } = Val,
948 <<
949     TXFRS:1, TXPHS:1, TXPRS:1, TXFRB:1, AAT:1, ESYNCR:1, CPLOCK:1, IRQS:1, %
950     bits 7-0
951     RXFCE:1, RXFCG:1, RXDFR:1, RXPHE:1, RXPHD:1, LDEDONE:1, RXSFDD:1, RXPRD:1,
952     % bits 15-8
953     SPL2INIT:1, GPIOIRQ:1, RXPTO:1, RXOVR:1, Reserved0:1, LDEERR:1, RXRFTO:1,
954     RXRFSL:1, % bits 23-16
955     ICRBP:1, HSRBP:1, AFFREJ:1, TXBERR:1, HPDWARN:1, RXSFDTO:1, CLCKPLL_LL:1,
956     RFPLL_LL:1, % bits 31-24
957     Reserved1:5, TXPUTE:1, RXPRES:1, RXRSCS:1 % bits 39-32
958 >>;
959 reg(decode, rx_finfo, Resp) ->
960 <<
961     RXPACC:12, RXPSR:2, RXPRFR:2, RNG:1, RXBR:2, RXNSPL:2, _:1, RXFLE:3,
962     RXFLEN:7
963 >> = reverse(Resp),
964 # {
965     rxpacc => RXPACC, rxpsr => RXPSR, rxprfr => RXPRFR, rng => RNG, rxbr =>
966     RXBR, rxnspl => RXNSPL, rxfle => RXFLE, rxflen => RXFLEN
967 }
968 reg(decode, rx_buffer, Resp) ->
969 # { rx_buffer => reverse(Resp)};
970 reg(decode, rx_fqual, Resp) ->
971 <<
972     CIR_PWR:16, PP_AMPL3:16, FP_AMPL2:16, STD_NOISE:16
973 >> = Resp,
974 # {

```



```

953     cir_pwr => CIR_PWR, pp_ampl3 => PP_AMPL3, fp_ampl2 => FP_AMPL2, std_noise
          => STD_NOISE
954 };
955 reg(decode, rx_ttcki, Resp) ->
956 <<
957     RXTTCKI:32
958 >> = reverse(Resp),
959 #{
960     rxttcki => RXTTCKI
961 };
962 reg(decode, rx_ttcko, Resp) ->
963 <<
964     _:1, RCPHASE:7, RSMDEL:8, _:5, RXTOFS:19
965 >> = reverse(Resp),
966 #{
967     rcphase => RCPHASE, rsmdel => RSMDEL, rxtofs => RXTOFS
968 };
969 reg(decode, rx_time, Resp) ->
970 <<
971     RX_RAWST:40, FP_AMPL1:16, FP_INDEX:16, RX_STAMP:40
972 >> = reverse(Resp),
973 #{
974     rx_rawst => RX_RAWST, fp_ampl1 => FP_AMPL1, fp_index => FP_INDEX, rx_stamp
          => RX_STAMP
975 };
976 reg(decode, tx_time, Resp) ->
977 <<
978     TX_RAWST:40, TX_STAMP:40
979 >> = reverse(Resp),
980 #{
981     tx_rawst => TX_RAWST, tx_stamp => TX_STAMP
982 };
983 reg(decode, tx_antd, Resp) ->
984 #{
985     tx_antd => reverse(Resp)
986 };
987 reg(encode, tx_antd, Val) ->
988 #{
989     tx_antd := TX_ANTD
990 } = Val,
991 reverse(<<
992     TX_ANTD:16
993 >>);
994 reg(decode, sys_state, Resp) ->
995 <<
996     _:8, _:4, PMSC_STATE:4, _:3, RX_STATE:5, _:4, TX_STATE:4
997 >> = reverse(Resp),
998 #{
999     pmsc_state => PMSC_STATE, rx_state => RX_STATE, tx_state => TX_STATE
1000 };
1001 reg(decode, ack_resp_t, Resp) ->
1002 <<
1003     ACK_TIME:8, _:4, W4R_TIME:20
1004 >> = reverse(Resp),
1005 #{
1006     ack_tim => ACK_TIME, w4r_tim => W4R_TIME
1007 };
1008 reg(encode, ack_resp_t, Val) ->
1009 #{
1010     ack_tim := ACK_TIME, w4r_tim := W4R_TIME
1011 } = Val,
1012 reverse(<<

```

```

1013     ACK_TIME:8, 2#0:4, W4R_TIME:20
1014     >>);
1015 reg(decode, rx_sniff, Resp) ->
1016     <<
1017         Reserved0:16, SNIFF_OFFT:8, Reserved1:4, SNIFF_ONT:4
1018     >> = reverse(Resp),
1019     #{
1020         res0 => Reserved0,
1021         sniff_offt => SNIFF_OFFT,
1022         sniff_ont => SNIFF_ONT,
1023         res1 => Reserved1
1024     };
1025 reg(encode, rx_sniff, Val) ->
1026     #{
1027         res0 := Reserved0,
1028         sniff_offt := SNIFF_OFFT,
1029         sniff_ont := SNIFF_ONT,
1030         res1 := Reserved1
1031     } = Val,
1032     reverse(<<
1033         Reserved0:16, SNIFF_OFFT:8, Reserved1:4, SNIFF_ONT:4
1034     >>);
1035 % Smart transmit power control (cf. user manual p 104)
1036 reg(decode, tx_power, Resp) ->
1037     <<
1038         BOOSTP125:8, BOOSTP250:8, BOOSTP500:8, BOOSTNORM:8
1039     >> = reverse(Resp),
1040     #{
1041         boostp125 => BOOSTP125, boostp250 => BOOSTP250, boostp500 => BOOSTP500,
1042         boostnorm => BOOSTNORM
1043     };
1044 % Leave the possibility to the user to write the value as one
1045 case Val of
1046     #{ tx_power := ValToEncode } -> reverse(<<ValToEncode:32>>);
1047     #{ boostp125 := BOOSTP125, boostp250 := BOOSTP250, boostp500 := BOOSTP500,
1048         boostnorm := BOOSTNORM } ->reverse(<<BOOSTP125:8, BOOSTP250:8,
1049         BOOSTP500:8, BOOSTNORM:8>>)
1050 end;
1051 reg(decode, chan_ctrl, Resp) ->
1052     <<
1053         RX_PCODE:5, TX_PCODE:5, RNSSFD:1, TNSSFD:1, RXPRF:2, DWSFD:1, Reserved0:9,
1054         RX_CHAN:4, TX_CHAN:4
1055     >> = reverse(Resp),
1056     #{
1057         rx_pcode => RX_PCODE, tx_pcode => TX_PCODE, rnssfd => RNSSFD, tnssfd =>
1058         TNSSFD, rxprf => RXPRF, dwsfd => DWSFD, res0 => Reserved0, rx_chan =>
1059         RX_CHAN, tx_chan => TX_CHAN
1060     };
1061 reg(encode, chan_ctrl, Val) ->
1062     #{
1063         rx_pcode := RX_PCODE, tx_pcode := TX_PCODE, rnssfd := RNSSFD, tnssfd :=
1064         TNSSFD, rxprf := RXPRF, dwsfd := DWSFD, res0 := Reserved0, rx_chan :=
1065         RX_CHAN, tx_chan := TX_CHAN
1066     } = Val,
1067     reverse(<<
1068         RX_PCODE:5, TX_PCODE:5, RNSSFD:1, TNSSFD:1, RXPRF:2, DWSFD:1, Reserved0:9,
1069         RX_CHAN:4, TX_CHAN:4
1070     >>);
1071 reg(encode, usr_sfd, Value) ->
1072     #{
1073         usr_sfd := USR_SFD

```

```

1066     } = Value,
1067     reverse(<<
1068         USR_SFD:(8*41)
1069     >>);
1070 reg(decode, usr_sfd, Resp) ->
1071     <<
1072         USR_SFD:(8*41)
1073     >> = reverse(Resp),
1074     #{
1075         usr_sfd => USR_SFD
1076     };
1077 % AGC_CTRL is a complex register with reserved bits that can't be written
1078 reg(encode, agc_ctrl1, Val) ->
1079     #{
1080         res := Reserved, dis_am := DIS_AM
1081     } = Val,
1082     reverse(<<
1083         Reserved:15, DIS_AM:1
1084     >>);
1085 reg(encode, agc_tune1, Val) ->
1086     #{
1087         agc_tune1 := AGC_TUNE1
1088     } = Val,
1089     reverse(<<
1090         AGC_TUNE1:16
1091     >>);
1092 reg(encode, agc_tune2, Val) ->
1093     #{
1094         agc_tune2 := AGC_TUNE2
1095     } = Val,
1096     reverse(<<
1097         AGC_TUNE2:32
1098     >>);
1099 reg(encode, agc_tune3, Val) ->
1100     #{
1101         agc_tune3 := AGC_TUNE3
1102     } = Val,
1103     reverse(<<
1104         AGC_TUNE3:16
1105     >>);
1106 reg(decode, agc_ctrl, Resp) ->
1107     <<
1108         _:4, EDV2:9, EDG1:5, _:6, % AGC_STAT1 (RP => don't save reserved bits)
1109         _:80, % Reserved 4
1110         AGC_TUNE3:16, % AGC_TUNE3
1111         _:16, % Reserved 3
1112         AGC_TUNE2:32, % AGC_TUNE2
1113         _:48, % Reserved 2
1114         AGC_TUNE1:16, % AGC_TUNE1
1115         Reserved0:15, DIS_AM:1, % AGC_CTRL1 (RW => save reserved bits)
1116         _:16 % Reserved 1
1117     >> = reverse(Resp),
1118     #{
1119         agc_ctrl1 => #{res => Reserved0, dis_am => DIS_AM},
1120         agc_tune1 => AGC_TUNE1,
1121         agc_tune2 => AGC_TUNE2,
1122         agc_tune3 => AGC_TUNE3,
1123         agc_stat1 => #{edv2 => EDV2, edg1 => EDG1}
1124     };
1125 reg(encode, ec_ctrl, Val) ->
1126     #{
1127         res := Reserved, ostrm := OSTRM, wait := WAIT, pll1dt := PLL1DT, osrsm :=

```

```

1128     OSRSM, ostsm := OSTSM
1129   } = Val,
1130   reverse(<<
1131     Reserved:20, OSTRM:1, WAIT:8, PLLLDT:1, OSRSM:1, OSTSM:1 % EC_CTRL
1132   >>);
1133 reg(decode, ext_sync, Resp) ->
1134   <<
1135     _:26, OFFSET_EXT:6, % EC_GLOP
1136     RX_TS_EST:32, % EC_RXTC
1137     Reserved:20, OSTRM:1, WAIT:8, PLLLDT:1, OSRSM:1, OSTSM:1 % EC_CTRL
1138   >> = reverse(Resp),
1139   #{
1140     ec_ctrl => #{res => Reserved, ostrm => OSTRM, wait => WAIT, pllldt =>
1141       PLLLDT, osrsm => OSRSM, ostsm => OSTSM},
1142     rx_ts_est => RX_TS_EST,
1143     ec_golp => #{offset_ext => OFFSET_EXT}
1144   };
1145 % "The host system doesn't need to access the ACC_MEM in normal operation, however
1146   it may be of interest [...] for diagnostic purpose" (from DW1000 user manual)
1147 reg(decode, acc_mem, Resp) ->
1148   #{
1149     acc_mem => reverse(Resp)
1150   };
1151 reg(encode, gpio_mode, Val) ->
1152   #{
1153     msgp8 := MSGP8, msgp7 := MSGP7, msgp6 := MSGP6, msgp5 := MSGP5, msgp4 :=
1154     MSGP4, msgp3 := MSGP3, msgp2 := MSGP2, msgp1 := MSGP1, msgp0 := MSGP0
1155   } = Val,
1156   reverse(<<
1157     2#0:8, MSGP8:2, MSGP7:2, MSGP6:2, MSGP5:2, MSGP4:2, MSGP3:2, MSGP2:2,
1158     MSGP1:2, MSGP0:2, 2#0:6 % GPIO_MODE
1159   >>);
1160 reg(encode, gpio_dir, Val) ->
1161   #{
1162     gdm8 := GDM8, gdm7 := GDM7, gdm6 := GDM6, gdm5 := GDM5, gdm4 := GDM4, gdm3
1163     := GDM3, gdm2 := GDM2, gdm1 := GDM1, gdm0 := GDM0,
1164     gdp8 := GDP8, gdp7 := GDP7, gdp6 := GDP6, gdp5 := GDP5, gdp4 := GDP4, gdp3
1165     := GDP3, gdp2 := GDP2, gdp1 := GDP1, gdp0 := GDP0
1166   } = Val,
1167   reverse(<<
1168     2#0:11, GDM8:1, 2#0:3, GDP8:1, GDM7:1, GDM6:1, GDM5:1, GDM4:1, GDP7:1,
1169     GDP6:1, GDP5:1, GDP4:1, GDM3:1, GDM2:1, GDM1:1, GDM0:1, GDP3:1, GDP2
1170     :1, GDP1:1, GDP0:1 % GPIO2_DIR
1171   >>);
1172 reg(encode, gpio_dout, Val) ->
1173   #{
1174     gom8 := GOM8, gom7 := GOM7, gom6 := GOM6, gom5 := GOM5, gom4 := GOM4, gom3
1175     := GOM3, gom2 := GOM2, gom1 := GOM1, gom0 := GOM0,
1176     gop8 := GOP8, gop7 := GOP7, gop6 := GOP6, gop5 := GOP5, gop4 := GOP4, gop3
1177     := GOP3, gop2 := GOP2, gop1 := GOP1, gop0 := GOP0
1178   } = Val,
1179   reverse(<<
1180     2#0:11, GOM8:1, 2#0:3, GOP8:1, GOM7:1, GOM6:1, GOM5:1, GOM4:1, GOP7:1,
1181     GOP6:1, GOP5:1, GOP4:1, GOM3:1, GOM2:1, GOM1:1, GOM0:1, GOP3:1, GOP2
1182     :1, GOP1:1, GOP0:1 % GPIO_DOUT
1183   >>);
1184 reg(encode, gpio_irqe, Val) ->
1185   #{
1186     girqe8 := GIRQE8, girqe7 := GIRQE7, girqe6 := GIRQE6, girqe5 := GIRQE5,
1187     girqe4 := GIRQE4, girqe3 := GIRQE3, girqe2 := GIRQE2, girqe1 := GIRQE1
1188     , girqe0 := GIRQE0
1189   } = Val,

```

```

1175     reverse(<<
1176         2#0:23, GIRQE8:1, GIRQE7:1, GIRQE6:1, GIRQE5:1, GIRQE4:1, GIRQE3:1, GIRQE2
           :1, GIRQE1:1, GIRQE0:1 % GPIO_IRQE
1177     >>);
1178 reg(encode, gpio_isen, Val) ->
1179     #{
1180         gisen8 := GISEN8, gisen7 := GISEN7, gisen6 := GISEN6, gisen5 := GISEN5,
           gisen4 := GISEN4, gisen3 := GISEN3, gisen2 := GISEN2, gisen1 := GISEN1
           , gisen0 := GISEN0
1181     } = Val,
1182     reverse(<<
1183         2#0:23, GISEN8:1, GISEN7:1, GISEN6:1, GISEN5:1, GISEN4:1, GISEN3:1, GISEN2
           :1, GISEN1:1, GISEN0:1 % GPIO_ISEN
1184     >>);
1185 reg(encode, gpio_imod, Val) ->
1186     #{
1187         gimod8 := GIMOD8, gimod7 := GIMOD7, gimod6 := GIMOD6, gimod5 := GIMOD5,
           gimod4 := GIMOD4, gimod3 := GIMOD3, gimod2 := GIMOD2, gimod1 := GIMOD1
           , gimod0 := GIMOD0
1188     } = Val,
1189     reverse(<<
1190         2#0:23, GIMOD8:1, GIMOD7:1, GIMOD6:1, GIMOD5:1, GIMOD4:1, GIMOD3:1, GIMOD2
           :1, GIMOD1:1, GIMOD0:1 % GPIO_IMOD
1191     >>);
1192 reg(encode, gpio_ibes, Val) ->
1193     #{
1194         gibes8 := GIBES8, gibes7 := GIBES7, gibes6 := GIBES6, gibes5 := GIBES5,
           gibes4 := GIBES4, gibes3 := GIBES3, gibes2 := GIBES2, gibes1 := GIBES1
           , gibes0 := GIBES0
1195     } = Val,
1196     reverse(<<
1197         2#0:23, GIBES8:1, GIBES7:1, GIBES6:1, GIBES5:1, GIBES4:1, GIBES3:1, GIBES2
           :1, GIBES1:1, GIBES0:1 % GPIO_IBES
1198     >>);
1199 reg(encode, gpio_iclr, Val) ->
1200     #{
1201         giclr8 := GICLR8, giclr7 := GICLR7, giclr6 := GICLR6, giclr5 := GICLR5,
           giclr4 := GICLR4, giclr3 := GICLR3, giclr2 := GICLR2, giclr1 := GICLR1
           , giclr0 := GICLR0
1202     } = Val,
1203     reverse(<<
1204         2#0:23, GICLR8:1, GICLR7:1, GICLR6:1, GICLR5:1, GICLR4:1, GICLR3:1, GICLR2
           :1, GICLR1:1, GICLR0:1 % GPIO_ICLR
1205     >>);
1206 reg(encode, gpio_idbe, Val) ->
1207     #{
1208         gidbe8 := GIDBE8, gidbe7 := GIDBE7, gidbe6 := GIDBE6, gidbe5 := GIDBE5,
           gidbe4 := GIDBE4, gidbe3 := GIDBE3, gidbe2 := GIDBE2, gidbe1 := GIDBE1
           , gidbe0 := GIDBE0
1209     } = Val,
1210     reverse(<<
1211         2#0:23, GIDBE8:1, GIDBE7:1, GIDBE6:1, GIDBE5:1, GIDBE4:1, GIDBE3:1, GIDBE2
           :1, GIDBE1:1, GIDBE0:1 % GPIO_IDBE
1212     >>);
1213 reg(encode, gpio_raw, Val) ->
1214     #{
1215         grawp8 := GRAWP8, grawp7 := GRAWP7, grawp6 := GRAWP6, grawp5 := GRAWP5,
           grawp4 := GRAWP4, grawp3 := GRAWP3, grawp2 := GRAWP2, grawp1 := GRAWP1
           , grawp0 := GRAWP0
1216     } = Val,
1217     reverse(<<
1218         2#0:23, GRAWP8:1, GRAWP7:1, GRAWP6:1, GRAWP5:1, GRAWP4:1, GRAWP3:1, GRAWP2

```

```

1219         :1, GRAWP1:1, GRAWP0:1 % GPIO_RAW
1220     >>);
1221     reg(decode, gpio_ctrl, Resp) ->
1222     <<
1223         _:23, GRAWP8:1, GRAWP7:1, GRAWP6:1, GRAWP5:1, GRAWP4:1, GRAWP3:1, GRAWP2
1224         :1, GRAWP1:1, GRAWP0:1, % GPIO_RAW
1225         _:23, GIDBE8:1, GIDBE7:1, GIDBE6:1, GIDBE5:1, GIDBE4:1, GIDBE3:1, GIDBE2
1226         :1, GIDBE1:1, GIDBE0:1, % GPIO_IDBE
1227         _:23, GICLR8:1, GICLR7:1, GICLR6:1, GICLR5:1, GICLR4:1, GICLR3:1, GICLR2
1228         :1, GICLR1:1, GICLR0:1, % GPIO_ICLR
1229         _:23, GIBES8:1, GIBES7:1, GIBES6:1, GIBES5:1, GIBES4:1, GIBES3:1, GIBES2
1230         :1, GIBES1:1, GIBES0:1, % GPIO_IBES
1231         _:23, GIMOD8:1, GIMOD7:1, GIMOD6:1, GIMOD5:1, GIMOD4:1, GIMOD3:1, GIMOD2
1232         :1, GIMOD1:1, GIMOD0:1, % GPIO_IMOD
1233         _:23, GISEN8:1, GISEN7:1, GISEN6:1, GISEN5:1, GISEN4:1, GISEN3:1, GISEN2
1234         :1, GISEN1:1, GISEN0:1, % GPIO_ISEN
1235         _:23, GIRQE8:1, GIRQE7:1, GIRQE6:1, GIRQE5:1, GIRQE4:1, GIRQE3:1, GIRQE2
1236         :1, GIRQE1:1, GIRQE0:1, % GPIO_IRQE
1237         _:11, GOM8:1, _:3, GOP8:1, GOM7:1, GOM6:1, GOM5:1, GOM4:1, GOP7:1, GOP6:1,
1238         GOP5:1, GOP4:1, GOM3:1, GOM2:1, GOM1:1, GOM0:1, GOP3:1, GOP2:1, GOP1
1239         :1, GOP0:1, % GPIO_DOUT
1240         _:11, GDM8:1, _:3, GDP8:1, GDM7:1, GDM6:1, GDM5:1, GDM4:1, GDP7:1, GDP6:1,
1241         GDP5:1, GDP4:1, GDM3:1, GDM2:1, GDM1:1, GDM0:1, GDP3:1, GDP2:1, GDP1
1242         :1, GDP0:1, % GPIO_DIR
1243         _:32, % Reserved
1244         _:8, MSGP8:2, MSGP7:2, MSGP6:2, MSGP5:2, MSGP4:2, MSGP3:2, MSGP2:2, MSGP1
1245         :2, MSGP0:2, _:6 % GPIO_MODE
1246     >> = reverse(Resp),
1247     #{
1248         gpio_mode => #{msgp8 => MSGP8, msgp7 => MSGP7, msgp6 => MSGP6, msgp5 =>
1249             MSGP5, msgp4 => MSGP4, msgp3 => MSGP3, msgp2 => MSGP2, msgp1 => MSGP1,
1250             msgp0 => MSGP0},
1251         gpio_dir => #{gdm8 => GDM8, gdm7 => GDM7, gdm6 => GDM6, gdm5 => GDM5, gdm4
1252             => GDM4, gdm3 => GDM3, gdm2 => GDM2, gdm1 => GDM1, gdm0 => GDM0,
1253             gdp8 => GDP8, gdp7 => GDP7, gdp6 => GDP6, gdp5 => GDP5, gdp4
1254             => GDP4, gdp3 => GDP3, gdp2 => GDP2, gdp1 => GDP1, gdp0
1255             => GDP0},
1256         gpio_dout => #{gom8 => GOM8, gom7 => GOM7, gom6 => GOM6, gom5 => GOM5,
1257             gom4 => GOM4, gom3 => GOM3, gom2 => GOM2, gom1 => GOM1, gom0 => GOM0,
1258             gop8 => GOP8, gop7 => GOP7, gop6 => GOP6, gop5 => GOP5,
1259             gop4 => GOP4, gop3 => GOP3, gop2 => GOP2, gop1 => GOP1,
1260             gop0 => GOP0},
1261         gpio_irqe => #{girqe8 => GIRQE8, girqe7 => GIRQE7, girqe6 => GIRQE6,
1262             girqe5 => GIRQE5, girqe4 => GIRQE4, girqe3 => GIRQE3, girqe2 => GIRQE2
1263             , girqe1 => GIRQE1, girqe0 => GIRQE0},
1264         gpio_isen => #{gisen8 => GISEN8, gisen7 => GISEN7, gisen6 => GISEN6,
1265             gisen5 => GISEN5, gisen4 => GISEN4, gisen3 => GISEN3, gisen2 => GISEN2
1266             , gisen1 => GISEN1, gisen0 => GISEN0},
1267         gpio_imod => #{gimod8 => GIMOD8, gimod7 => GIMOD7, gimod6 => GIMOD6,
1268             gimod5 => GIMOD5, gimod4 => GIMOD4, gimod3 => GIMOD3, gimod2 => GIMOD2
1269             , gimod1 => GIMOD1, gimod0 => GIMOD0},
1270         gpio_ibes => #{gibes8 => GIBES8, gibes7 => GIBES7, gibes6 => GIBES6,
1271             gibes5 => GIBES5, gibes4 => GIBES4, gibes3 => GIBES3, gibes2 => GIBES2
1272             , gibes1 => GIBES1, gibes0 => GIBES0},
1273         gpio_iclr => #{giclr8 => GICLR8, giclr7 => GICLR7, giclr6 => GICLR6,
1274             giclr5 => GICLR5, giclr4 => GICLR4, giclr3 => GICLR3, giclr2 => GICLR2
1275             , giclr1 => GICLR1, giclr0 => GICLR0},
1276         gpio_idbe => #{gidbe8 => GIDBE8, gidbe7 => GIDBE7, gidbe6 => GIDBE6,
1277             gidbe5 => GIDBE5, gidbe4 => GIDBE4, gidbe3 => GIDBE3, gidbe2 => GIDBE2
1278             , gidbe1 => GIDBE1, gidbe0 => GIDBE0},
1279         gpio_raw => #{grawp8 => GRAWP8, grawp7 => GRAWP7, grawp6 => GRAWP6, grawp5
1280             => GRAWP5, grawp4 => GRAWP4, grawp3 => GRAWP3, grawp2 => GRAWP2,

```

```

1247     };
1248 reg(encode, drx_tune0b, Val) ->
1249     #{
1250         drx_tune0b := DRX_TUNE0b
1251     } = Val,
1252     reverse(<<
1253         DRX_TUNE0b:16
1254     >>);
1255 reg(encode, drx_tune1a, Val) ->
1256     #{
1257         drx_tune1a := DRX_TUNE1a
1258     } = Val,
1259     reverse(<<
1260         DRX_TUNE1a:16
1261     >>);
1262 reg(encode, drx_tune1b, Val) ->
1263     #{
1264         drx_tune1b := DRX_TUNE1b
1265     } = Val,
1266     reverse(<<
1267         DRX_TUNE1b:16
1268     >>);
1269 reg(encode, drx_tune2, Val) ->
1270     #{
1271         drx_tune2 := DRX_TUNE2
1272     } = Val,
1273     reverse(<<
1274         DRX_TUNE2:32
1275     >>);
1276 reg(encode, drx_sfdtoc, Val) ->
1277     #{
1278         drx_sfdtoc := DRX_SFDTOC
1279     } = Val,
1280     reverse(<<
1281         DRX_SFDTOC:16
1282     >>);
1283 reg(encode, drx_pretoc, Val) ->
1284     #{
1285         drx_pretoc := DRX_PRETOC
1286     } = Val,
1287     reverse(<<
1288         DRX_PRETOC:16
1289     >>);
1290 reg(encode, drx_tune4h, Val) ->
1291     #{
1292         drx_tune4h := DRX_TUNE4H
1293     } = Val,
1294     reverse(<<
1295         DRX_TUNE4H:16
1296     >>);
1297 reg(decode, drx_conf, Resp) ->
1298     <<
1299         RXPACC_NOSAT:8, % present in the user manual but not in the driver code in
1300             C
1301         % _:8, % Placeholder for the remaining 8 bits
1302         DRX_CAR_INT:24,
1303         DRX_TUNE4H:16,
1304         DRX_PRETOC:16,
1305         _:16,
1306         DRX_SFDTOC:16,
1307         _:160,

```

```

1307     DRX_TUNE2:32,
1308     DRX_TUNE1b:16,
1309     DRX_TUNE1a:16,
1310     DRX_TUNE0b:16,
1311     _:16
1312 >> = reverse(Resp),
1313 #{
1314     drx_tune0b => DRX_TUNE0b,
1315     drx_tune1a => DRX_TUNE1a,
1316     drx_tune1b => DRX_TUNE1b,
1317     drx_tune2 => DRX_TUNE2,
1318     drx_tune4h => DRX_TUNE4H,
1319     drx_car_int => DRX_CAR_INT,
1320     drx_sfdtoc => DRX_SFDTOC,
1321     drx_pretoc => DRX_PRETOC,
1322     rxpacc_nosat => RXPACC_NOSAT
1323 };
1324 reg(encode, rf_conf, Val) ->
1325 #{
1326     txrxsw := TXRXSW, ldofen := LDOFEN, pllfen := PLLFEN, txfen := TXFEN
1327 } = Val,
1328 reverse(<<
1329     2#0:9, TXRXSW:2, LDOFEN:5, PLLFEN:3, TXFEN:5, 2#0:8 % RF_CONF
1330 >>);
1331 reg(encode, rf_rxctrlh, Val) ->
1332 #{
1333     rf_rxctrlh := RF_RXCTRLH
1334 } = Val,
1335 reverse(<<
1336     RF_RXCTRLH:8 % RF_RXCTRLH
1337 >>);
1338 % user manual gives fields but encoding should be done as one following table 38
1339 reg(encode, rf_txctrl, Val) ->
1340 #{
1341     rf_txctrl := RF_TXCTRL
1342 } = Val,
1343 reverse(<<
1344     RF_TXCTRL:32
1345 >>);
1346 reg(encode, ldotune, Val) ->
1347 #{
1348     ldotune := LDOTUNE
1349 } = Val,
1350 reverse(<<
1351     LDOTUNE:40
1352 >>);
1353 reg(decode, rf_conf, Resp) ->
1354 <<
1355     _:40, % Placeholder for the remaining 40 bits
1356     LDOTUNE:40, % LDOTUNE
1357     _:28, RFPLLLOCK:1, CPLLHIGH:1, CPLLLOW:1, CPLLLOCK:1, % RF_STATUS
1358     _:128, _:96, % Reserved 2 - On user manual 16 bytes but offset gives 28
        bytes (16 bytes (128 bits) + 12 bytes (96 bits))
1359     RF_TXCTRL:32, % cf. encode function: Reserved:20, TXMQ:3, TXMTUNE:4, _:5 -
        RF_TXCTRL
1360     RF_RXCTRLH:8, % RF_RXCTRLH
1361     _:56, % Reserved 1
1362     _:9, TXRXSW:2, LDOFEN:5, PLLFEN:3, TXFEN:5, _:8 % RF_CONF
1363 >> = reverse(Resp),
1364 #{
1365     ldotune => LDOTUNE,
1366     rf_status => #{rfplllock => RFPLLLOCK, cplllow => CPLLLOW, cpllhigh =>

```



```

        CPLLHIGH, cpplllock => CPLLLOCK},
1367     rf_txctrl => RF_TXCTRL,
1368     rf_rxctrlh => RF_RXCTRLH,
1369     rf_conf => #{txrxsw => TXRXSW, ldofen => LDOFEN, pllflen => PLLFEN, txfen
        => TXFEN}
1370 };
1371 reg(encode, tc_sarc, Val) ->
1372   #{
1373     sar_ctrl := SAR_CTRL
1374   } = Val,
1375   reverse(<<
1376     2#0:15, SAR_CTRL:1
1377   >>);
1378 reg(encode, tc_pg_ctrl, Val) ->
1379   #{
1380     pg_tmeas := PG_TMEAS, res := Reserved, pg_start := PG_START
1381   } = Val,
1382   reverse(<<
1383     2#0:2, PG_TMEAS:4, Reserved:1, PG_START:1
1384   >>);
1385 reg(encode, tc_pgdelay, Val) ->
1386   #{
1387     tc_pgdelay := TC_PGDELAY
1388   } = Val,
1389   reverse(<<
1390     TC_PGDELAY:8
1391   >>);
1392 reg(encode, tc_pgtest, Val) ->
1393   #{
1394     tc_pgtest := TC_PGTEST
1395   } = Val,
1396   reverse(<<
1397     TC_PGTEST:8
1398   >>);
1399 reg(decode, tx_cal, Resp) ->
1400   <<
1401     TC_PGTEST:8, % TC_PGTEST
1402     TC_PGDELAY:8, % TC_PGDELAY
1403     _:4, DELAY_CNT:12, % TC_PG_STATUS
1404     _:2, PG_TMEAS:4, Reserved0:1, PG_START:1, % TC_PG_CTRL
1405     SAR_WTEMP:8, SAR_WVBAT:8, % TC_SARW
1406     _:8, SAR_LTEMP:8, SAR_LVBAT:8, % TC_SARL
1407     _:8, % Place holder to fill the gap between the offsets
1408     _:15, SAR_CTRL:1 % TC_SARC
1409   >> = reverse(Resp),
1410   #{
1411     tc_pgtest => TC_PGTEST,
1412     tc_pgdelay => TC_PGDELAY,
1413     tc_pg_status => #{delay_cnt => DELAY_CNT},
1414     tc_pg_ctrl => #{pg_tmeas => PG_TMEAS, res => Reserved0, pg_start =>
        PG_START},
1415     tc_sarw => #{sar_wtemp => SAR_WTEMP, sar_wvbat => SAR_WVBAT},
1416     tc_sarl => #{sar_ltemp => SAR_LTEMP, sar_lvbat => SAR_LVBAT},
1417     tc_sarc => #{sar_ctrl => SAR_CTRL}
1418   };
1419 reg(encode, fs_pllcfg, Val) ->
1420   #{
1421     fs_pllcfg := FS_PLLCFG
1422   } = Val,
1423   reverse(<<
1424     FS_PLLCFG:32
1425   >>);

```

```

1426 reg(encode, fs_pll_tune, Val) ->
1427   #{
1428     fs_pll_tune := FS_PLLTUNE
1429   } = Val,
1430   reverse(<<
1431     FS_PLLTUNE:8
1432   >>);
1433 reg(encode, fs_xtalt, Val) ->
1434   #{
1435     res := Reserved, xtalt := XTALT
1436   } = Val,
1437   reverse(<<
1438     Reserved:3, XTALT:5
1439   >>);
1440 reg(decode, fs_ctrl, Resp) ->
1441   <<
1442     _:48, % Reserved 3
1443     Reserved:3, XTALT:5, % FS_XTALT
1444     _:16, % Reserved 2
1445     FS_PLLTUNE:8, % FS_PLLTUNE
1446     FS_PLLCFG:32, % FS_PLLCFG
1447     _:56 % Reserved 1
1448   >> = reverse(Resp),
1449   #{
1450     fs_xtalt => #{res => Reserved, xtalt => XTALT},
1451     fs_pll_tune => FS_PLLTUNE,
1452     fs_pll_cfg => FS_PLLCFG
1453   };
1454 reg(encode, aon_wcfg, Val) ->
1455   #{
1456     onw_lld := ONW_LLD, onw_llde := ONW_LLDE, pres_slee := PRES_SLEE, own_l64
1457       := OWN_L64, own_ldc := OWN_LDC, own_leui := OWN_LEUI, own_rx := OWN_RX
1458       , own_rad := OWN_RAD
1459   } = Val,
1460   reverse(<<
1461     2#0:3, ONW_LLD:1, ONW_LLDE:1, 2#0:2, PRES_SLEE:1, OWN_L64:1, OWN_LDC:1,
1462     2#0:2, OWN_LEUI:1, 2#0:1, OWN_RX:1, OWN_RAD:1 % AON_WCFG
1463   >>);
1464 reg(encode, aon_ctrl, Val) ->
1465   #{
1466     dca_enab := DCA_ENAB, dca_read := DCA_READ, upl_cfg := UPL_CFG, save :=
1467       SAVE, restore := RESTORE
1468   } = Val,
1469   reverse(<<
1470     DCA_ENAB:1, 2#0:3, DCA_READ:1, UPL_CFG:1, SAVE:1, RESTORE:1 % AON_CTRL
1471   >>);
1472 reg(encode, aon_rdat, Val) ->
1473   #{
1474     aon_rdat := AON_RDAT
1475   } = Val,
1476   reverse(<<
1477     AON_RDAT:8 % AON_RDAT
1478   >>);
1479 reg(encode, aon_addr, Val) ->
1480   #{
1481     aon_addr := AON_ADDR
1482   } = Val,
1483   reverse(<<
1484     AON_ADDR:8 % AON_ADDR
1485   >>);
1486 reg(encode, aon_cfg0, Val) ->
1487   #{

```

```

1484     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
1485     } = Val,
1486     reverse(<<
1487         SLEEP_TIM:16, LPCLKDIVA:11, LPDIV_EN:1, WAKE_CNT:1, WAKE_SPI:1, WAKE_PIN
            :1, SLEEP_EN:1 % AON_CFG0
1488     >>);
1489 reg(encode, aon_cfg1, Val) ->
1490     #{
1491         res := Reserved, lposc_c := LPOSC_C, smxx := SMXX, sleep_ce := SLEEP_CE
1492     } = Val,
1493     reverse(<<
1494         Reserved:13, LPOSC_C:1, SMXX:1, SLEEP_CE:1 % AON_CFG1
1495     >>);
1496 reg(decode, aon, Resp) ->
1497     <<
1498         Reserved:13, LPOSC_C:1, SMXX:1, SLEEP_CE:1, % AON_CFG1
1499         SLEEP_TIM:16, LPCLKDIVA:11, LPDIV_EN:1, WAKE_CNT:1, WAKE_SPI:1, WAKE_PIN
            :1, SLEEP_EN:1, % AON_CFG0
1500         _:8, % Reserved 1
1501         AON_ADDR:8, % AON_ADDR
1502         AON_RDAT:8, % AON_RDAT
1503         DCA_ENAB:1, _:3, DCA_READ:1, UPL_CFG:1, SAVE:1, RESTORE:1, % AON_CTRL
1504         _:3, ONW_LLD:1, ONW_LLDE:1, _:2, PRES_SLEE:1, ONW_L64:1, ONW_LDC:1, _:2,
            ONW_LEUI:1, _:1, ONW_RX:1, ONW_RAD:1 % AON_WCFG
1505     >> = reverse(Resp),
1506     #{
1507         aon_cfg1 => #{res => Reserved, lposc_c => LPOSC_C, smxx => SMXX, sleep_ce
            => SLEEP_CE},
1508         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},
1509         aon_addr => AON_ADDR,
1510         aon_rdat => AON_RDAT,
1511         aon_ctrl => #{dca_enab => DCA_ENAB, dca_read => DCA_READ, upl_cfg =>
            UPL_CFG, save => SAVE, restore => RESTORE},
1512         aon_wcfg => #{onw_lld => ONW_LLD, onw_llde => ONW_LLDE, pres_slee =>
            PRES_SLEE, onw_l64 => ONW_L64, onw_ldc => ONW_LDC, onw_leui =>
            ONW_LEUI, onw_rx => ONW_RX, onw_rad => ONW_RAD}
1513     };
1514 reg(encode, otp_wdat, Val) ->
1515     #{
1516         otp_wdat := OTP_WDAT
1517     } = Val,
1518     reverse(<<
1519         OTP_WDAT:32 % OTP_WDAT
1520     >>);
1521 reg(encode, otp_addr, Val) ->
1522     #{
1523         otpaddr := OTP_ADDR, res := Reserved
1524     } = Val,
1525     reverse(<<
1526         Reserved:5, OTP_ADDR:11 % OTP_ADDR
1527     >>);
1528 reg(encode, otp_ctrl, Val) ->
1529     #{
1530         ldload := LDELOAD, res1 := Reserved1, otpmr := OTPMR, otpprog := OTPPROG,
            res2 := Reserved2, otpmrwr := OTPMRWR, res3 := Reserved3, otpread :=
            OTPREAD, otp_rden := OTPRDEN
1531     } = Val,
1532     reverse(<<

```

```

1533     LDELOAD:1, Reserved1:4, OTPMR:4, OTPPROG:1, Reserved2:2, OTPMRWR:1,
        Reserved3:1, OTPREAD:1, OTPRDEN:1 % OTP_CTRL
1534     >>);
1535 reg(encode, otp_stat, Val) ->
1536     #{
1537         res := Reserved, otp_vpok := OTP_VPOK, otpprgd := OTPPRGD
1538     } = Val,
1539     reverse(<<
1540         Reserved:14, OTP_VPOK:1, OTPPRGD:1 % OTP_STAT
1541     >>);
1542 reg(encode, otp_rdat, Val) ->
1543     #{
1544         otp_rdat := OTP_RDAT
1545     } = Val,
1546     reverse(<<
1547         OTP_RDAT:32 % OTP_RDAT
1548     >>);
1549 reg(encode, opt_srdat, Val) ->
1550     #{
1551         otp_srdat := OTP_SRDAT
1552     } = Val,
1553     reverse(<<
1554         OTP_SRDAT:32 % OTP_SRDAT
1555     >>);
1556 reg(encode, otp_sf, Val) ->
1557     #{
1558         res1 := Reserved1, ops_sel := OPS_SEL, res2 := Reserved2, ldo_kick :=
            LDO_KICK, ops_kick := OPS_KICK
1559     } = Val,
1560     reverse(<<
1561         Reserved1:2, OPS_SEL:1, Reserved2:3, LDO_KICK:1, OPS_KICK:1 % OTP_SF
1562     >>);
1563 reg(decode, otp_if, Resp) ->
1564     <<
1565         Reserved5:2, OPS_SEL:1, Reserved6:3, LDO_KICK:1, OPS_KICK:1, % OTP_SF
1566         OTP_SRDAT:32, % OTP_SRDAT
1567         OTP_RDAT:32, % OTP_RDAT
1568         Reserved4:14, OTP_VPOK:1, OTPPRGD:1, % OTP_STAT
1569         LDELOAD:1, Reserved1:4, OTPMR:4, OTPPROG:1, Reserved2:2, OTPMRWR:1,
            Reserved3:1, OTPREAD:1, OTPRDEN:1, % OTP_CTRL
1570         Reserved0:5, OTP_ADDR:11, % OTP_ADDR
1571         OTP_WDAT:32 % OTP_WDAT
1572     >> = reverse(Resp),
1573     #{
1574         otp_sf => #{res1 => Reserved5, ops_sel => OPS_SEL, res2 => Reserved6,
            ldo_kick => LDO_KICK, ops_kick => OPS_KICK},
1575         otp_srdat => OTP_SRDAT,
1576         otp_rdat => OTP_RDAT,
1577         otp_stat => #{res => Reserved4, otp_vpok => OTP_VPOK, otpprgd => OTPPRGD},
1578         otp_ctrl => #{ldeload => LDELOAD, res1 => Reserved1, otpmr => OTPMR,
            otpprog => OTPPROG, res2 => Reserved2, otpmrwr => OTPMRWR, res3 =>
            Reserved3, otpread => OTPREAD, otp_rden => OTPRDEN},
1579         otp_addr => #{otpaddr => OTP_ADDR, res => Reserved0},
1580         otp_wdat => OTP_WDAT
1581     };
1582 reg(decode, lde_thresh, Resp) ->
1583     <<
1584         LDE_THRESH:16
1585     >> = reverse(Resp),
1586     #{
1587         lde_thresh => LDE_THRESH
1588     };

```

```

1589 reg(encode, lde_cfg1, Val) ->
1590   #{
1591     pmult := PMULT, ntm := NTM
1592   } = Val,
1593   reverse(<<
1594     PMULT:3, NTM:5
1595   >>);
1596 reg(decode, lde_cfg1, Resp) ->
1597   <<
1598     PMULT:3, NTM:5
1599   >> = reverse(Resp),
1600   #{
1601     lde_cfg1 => #{pmult => PMULT, ntm => NTM}
1602   };
1603 reg(decode, lde_ppindx, Resp) ->
1604   <<
1605     LDE_PPINDX:16
1606   >> = reverse(Resp),
1607   #{
1608     lde_ppindx => LDE_PPINDX
1609   };
1610 reg(decode, lde_ppampl, Resp) ->
1611   <<
1612     LDE_PPAMPL:16
1613   >> = reverse(Resp),
1614   #{
1615     lde_ppampl => LDE_PPAMPL
1616   };
1617 reg(encode, lde_rxantd, Val) ->
1618   #{
1619     lde_rxantd := LDE_RXANTD
1620   } = Val,
1621   reverse(<<
1622     LDE_RXANTD:16
1623   >>);
1624 reg(decode, lde_rxantd, Resp) ->
1625   <<
1626     LDE_RXANTD:16
1627   >> = reverse(Resp),
1628   #{
1629     lde_rxantd => LDE_RXANTD
1630   };
1631 reg(encode, lde_cfg2, Val) ->
1632   #{
1633     lde_cfg2 := LDE_CFG2
1634   } = Val,
1635   reverse(<<
1636     LDE_CFG2:16
1637   >>);
1638 reg(decode, lde_cfg2, Resp) ->
1639   <<
1640     LDE_CFG2:16
1641   >> = reverse(Resp),
1642   #{
1643     lde_cfg2 => LDE_CFG2
1644   };
1645 reg(encode, lde_repc, Val) ->
1646   #{
1647     lde_repc := LDE_REPC
1648   } = Val,
1649   reverse(<<
1650     LDE_REPC:16

```

```

1651     >>);
1652 reg(decode, lde_repc, Resp) ->
1653 <<
1654     LDE_REPC:16
1655     >> = reverse(Resp),
1656     #{
1657         lde_repc => LDE_REPC
1658     };
1659 reg(encode, evc_ctrl, Val) ->
1660     #{
1661         evc_clr := EVC_CLR, evc_en := EVC_EN
1662     } = Val,
1663     reverse(<<
1664         2#0:30, EVC_CLR:1, EVC_EN:1 % EVC_CTRL
1665     >>);
1666 reg(encode, diag_tmc, Val) ->
1667     #{
1668         tx_pstm := TX_PSTM
1669     } = Val,
1670     reverse(<<
1671         2#0:11, TX_PSTM:1, 2#0:4 % DIAG_TMC
1672     >>);
1673 reg(decode, dig_diag, Resp) ->
1674 <<
1675     _:11, TX_PSTM:1, _:4, % DIAG_TMC
1676     _:64, % Reserved 1
1677     _:4, EVC_TPW:12, % EVC_TPW
1678     _:4, EVC_HPW:12, % EVC_HPW
1679     _:4, EVC_TXFS:12, % EVC_TXFS
1680     _:4, EVC_FWTO:12, % EVC_FWTO
1681     _:4, EVC_PTO:12, % EVC_PTO
1682     _:4, EVC_STO:12, % EVC_STO
1683     _:4, ECV_OVR:12, % EVC_OVR
1684     _:4, EVC_FFR:12, % EVC_FFR
1685     _:4, EVC_FCE:12, % EVC_FCE
1686     _:4, EVC_FCG:12, % EVC_FCG
1687     _:4, EVC_RSE:12, % EVC_RSE
1688     _:4, EVC_PHE:12, % EVC_PHE
1689     _:30, EVC_CLR:1, EVC_EN:1 % EVC_CTRL
1690 >> = reverse(Resp),
1691     #{
1692         diag_tmc => #{tx_pstm => TX_PSTM},
1693         evc_tpw => EVC_TPW,
1694         evc_hpw => EVC_HPW,
1695         evc_txfs => EVC_TXFS,
1696         evc_fwto => EVC_FWTO,
1697         evc_pto => EVC_PTO,
1698         evc_sto => EVC_STO,
1699         evc_ovr => ECV_OVR,
1700         evc_ffr => EVC_FFR,
1701         evc_fce => EVC_FCE,
1702         evc_fcg => EVC_FCG,
1703         evc_rse => EVC_RSE,
1704         evc_phe => EVC_PHE,
1705         evc_ctrl => #{evc_clr => EVC_CLR, evc_en => EVC_EN}
1706     };
1707 reg(encode, pmsc_ctrl0, Val) ->
1708     #{
1709         softreset := SOFTRESET, pll2_seq_en := PLL2_SEQ_EN, khzclken := KHZCLKEN,
1710         gpdrn := GPDRN, gpdce := GPDCE,
1711         gprn := GPRN, gpce := GPCE, amce := AMCE, adcce := ADCCE, otp := OTP, res8
1712         := 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
1711     } = Val,
1712     reverse(<<
1713         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
1714     >>);
1715 reg(encode, pmsc_ctrl1, Val) ->
1716     #{
1717         khzclkdiv := KHZCLKDIV, lderune := LDERUNE, pllsyn := PLLSYN, snozr :=
            SNOZR, snoze := SNOZE, arxslp := ARXSLP, atxslp := ATXSLP, pktseq :=
            PKTSEQ, arx2init := ARX2INIT
1718     } = Val,
1719     reverse(<<
1720         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
1721     >>);
1722 reg(encode, pmsc_snozt, Val) ->
1723     #{
1724         snoz_tim := SNOZ_TIM
1725     } = Val,
1726     reverse(<<
1727         SNOZ_TIM:8 % PMSC_SNOZT
1728     >>);
1729 reg(encode, pmsc_txfseq, Val) ->
1730     #{
1731         txfineseq := TXFINESEQ
1732     } = Val,
1733     reverse(<<
1734         TXFINESEQ:16 % PMSC_TXFINESEQ
1735     >>);
1736 reg(encode, pmsc_ledc, Val) ->
1737     #{
1738         res31 := RES31, blnknow := BLNKNOW, res15 := RES15, blnken := BLNKEN,
            blink_tim := BLINK_TIM
1739     } = Val,
1740     reverse(<<
1741         RES31:12, BLNKNOW:4, RES15:7, BLNKEN:1, BLINK_TIM:8 % PMSC_LEDC
1742     >>);
1743 % mapping pmsc ctrl0 from: https://forum.qorvo.com/t/pmsc-ctrl0-bits8-15/746/3
1744 reg(decode, pmsc, Resp) ->
1745     % User manual says: reserved bits should be preserved at their reset value =>
        % can hardcode their values ? Safe to do that ?
1746     <<
1747         Res31:12, BLNKNOW:4, Res15:7, BLNKEN:1, BLINK_TIM:8, % PMSC_LEDC
1748         TXFINESEQ:16, % PMSC_TXFINESEQ
1749         _:(25*8), % Reserved 2
1750         SNOZ_TIM:8, % PMSC_SNOZT
1751         _:32, % Reserved 1
1752         KHZCLKDIV:6, _:8, LDERUNE:1, _:1, PLLSYN:1, SNOZR:1, SNOZE:1, ARXSLP:1,
            ATXSLP:1, PKTSEQ:8, _:1, ARX2INIT:1, _:1, % PMSC_CTRL1
1753         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
1754     >> = reverse(Resp),
1755     #{
1756         pmsc_ledc => #{res31 => Res31, blnknow => BLNKNOW, res15 => Res15, blnken
            => BLNKEN, blink_tim => BLINK_TIM},
1757         pmsc_txfseq => #{txfineseq => TXFINESEQ},
1758         pmsc_snozt => #{snoz_tim => SNOZ_TIM},
1759         pmsc_ctrl1 => #{khzclkdiv => KHZCLKDIV, lderune => LDERUNE, pllsyn =>

```

```

        PLLSYN, snozr => SNOZR, snoze => SNOZE, arxslp => ARXSLEP, atxslp =>
        ATXSLEP, pktseq => PKTSEQ, arx2init => ARX2INIT},
1760 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}
1761 };
1762 reg(decode, RegFile, Resp) -> error({unknown_regfile_to_decode, RegFile, Resp});
1763 reg(encode, RegFile, Resp) -> error({unknown_regfile_to_encode, RegFile, Resp}).
1764
1765 rw(read) -> 0;
1766 rw(write) -> 1.
1767
1768 % Mapping of the different register IDs to their hexadecimal value
1769 regFile(dev_id) -> 16#00;
1770 regFile(eui) -> 16#01;
1771 % 0x02 is reserved
1772 regFile(panadr) -> 16#03;
1773 regFile(sys_cfg) -> 16#04;
1774 % 0x05 is reserved
1775 regFile(sys_time) -> 16#06;
1776 % 0x07 is reserved
1777 regFile(tx_fctrl) -> 16#08;
1778 regFile(tx_buffer) -> 16#09;
1779 regFile(dx_time) -> 16#0A;
1780 % 0x0B is reserved
1781 regFile(rx_fwto) -> 16#0C;
1782 regFile(sys_ctrl) -> 16#0D;
1783 regFile(sys_mask) -> 16#0E;
1784 regFile(sys_status) -> 16#0F;
1785 regFile(rx_finfo) -> 16#10;
1786 regFile(rx_buffer) -> 16#11;
1787 regFile(rx_fqual) -> 16#12;
1788 regFile(rx_ttcki) -> 16#13;
1789 regFile(rx_ttcko) -> 16#14;
1790 regFile(rx_time) -> 16#15;
1791 % 0x16 is reserved
1792 regFile(tx_time) -> 16#17;
1793 regFile(tx_antd) -> 16#18;
1794 regFile(sys_state) -> 16#19;
1795 regFile(ack_resp_t) -> 16#1A;
1796 % 0x1B is reserved
1797 % 0x1C is reserved
1798 regFile(rx_sniff) -> 16#1D;
1799 regFile(tx_power) -> 16#1E;
1800 regFile(chan_ctrl) -> 16#1F;
1801 % 0x20 is reserved
1802 regFile(usr_sfd) -> 16#21;
1803 % 0x22 is reserved
1804 regFile(agc_ctrl) -> 16#23;
1805 regFile(ext_sync) -> 16#24;
1806 regFile(acc_mem) -> 16#25;
1807 regFile(gpio_ctrl) -> 16#26;
1808 regFile(drx_conf) -> 16#27;
1809 regFile(rf_conf) -> 16#28;
1810 % 0x29 is reserved
1811 regFile(tx_cal) -> 16#2A;
1812 regFile(fs_ctrl) -> 16#2B;
1813 regFile(aon) -> 16#2C;
1814 regFile(otp_if) -> 16#2D;
1815 regFile(lde_ctrl) -> regFile(lde_if); % No size ?

```



```

1816 regFile(lde_if) -> 16#2E;
1817 regFile(dig_diag) -> 16#2F;
1818 % 0x30 - 0x35 are reserved
1819 regFile(pmsc) -> 16#36;
1820 % 0x37 - 0x3F are reserved
1821 regFile(RegId) -> error({wrong_register_ID, RegId}).
1822
1823 % Only the writable subregisters in SRW register files are present here
1824 % AGC_CTRL
1825 subReg(agc_ctrl1) -> 16#02;
1826 subReg(agc_tune1) -> 16#04;
1827 subReg(agc_tune2) -> 16#0C;
1828 subReg(agc_tune3) -> 16#12;
1829 subReg(agc_stat1) -> 16#1E;
1830 subReg(ec_ctrl) -> 16#00;
1831 subReg(gpio_mode) -> 16#00;
1832 subReg(gpio_dir) -> 16#08;
1833 subReg(gpio_dout) -> 16#0C;
1834 subReg(gpio_irqe) -> 16#10;
1835 subReg(gpio_isen) -> 16#14;
1836 subReg(gpio_imode) -> 16#18;
1837 subReg(gpio_ibes) -> 16#1C;
1838 subReg(gpio_iclr) -> 16#20;
1839 subReg(gpio_idbe) -> 16#24;
1840 subReg(gpio_raw) -> 16#28;
1841 subReg(drx_tune0b) -> 16#02;
1842 subReg(drx_tune1a) -> 16#04;
1843 subReg(drx_tune1b) -> 16#06;
1844 subReg(drx_tune2) -> 16#08;
1845 subReg(drx_sfdtoc) -> 16#20;
1846 subReg(drx_pretoc) -> 16#24;
1847 subReg(drx_tune4h) -> 16#26;
1848 subReg(rf_conf) -> 16#00;
1849 subReg(rf_rxctrlh) -> 16#0B;
1850 subReg(rf_txctrl) -> 16#0C;
1851 subReg(ldotune) -> 16#30;
1852 subReg(tc_sarc) -> 16#00;
1853 subReg(tc_pg_ctrl) -> 16#08;
1854 subReg(tc_pgdelay) -> 16#0B;
1855 subReg(tc_pgtest) -> 16#0C;
1856 subReg(fs_pllcfg) -> 16#07;
1857 subReg(fs_plltune) -> 16#0B;
1858 subReg(fs_xtalt) -> 16#0E;
1859 subReg(aon_wcfg) -> 16#00;
1860 subReg(aon_ctrl) -> 16#02;
1861 subReg(aon_rdat) -> 16#03;
1862 subReg(aon_addr) -> 16#04;
1863 subReg(aon_cfg0) -> 16#06;
1864 subReg(aon_cfg1) -> 16#0A;
1865 subReg(otp_wdat) -> 16#00;
1866 subReg(otp_addr) -> 16#04;
1867 subReg(otp_ctrl) -> 16#06;
1868 subReg(otp_stat) -> 16#08;
1869 subReg(otp_rdat) -> 16#0A;
1870 subReg(otp_srdat) -> 16#0E;
1871 subReg(otp_sf) -> 16#12;
1872 subReg(lde_thresh) -> 16#00;
1873 subReg(lde_cfg1) -> 16#806;
1874 subReg(lde_ppindx) -> 16#1000;
1875 subReg(lde_ppampl) -> 16#1002;
1876 subReg(lde_rxantd) -> 16#1804;
1877 subReg(lde_cfg2) -> 16#1806;

```

```

1878 subReg(lde_repc) -> 16#2804;
1879 subReg(evc_ctrl) -> 16#00;
1880 subReg(diag_tmc) -> 16#24;
1881 subReg(pmsc_ctrl0) -> 16#00;
1882 subReg(pmsc_ctrl1) -> 16#04;
1883 subReg(pmsc_snozt) -> 16#0C;
1884 subReg(pmsc_txfseq) -> 16#26;
1885 subReg(pmsc_ledc) -> 16#28.
1886
1887
1888 % Mapping of the size in bytes of the different register IDs
1889 regSize(dev_id) -> 4;
1890 regSize(eui) -> 8;
1891 regSize(panadr) -> 4;
1892 regSize(sys_cfg) -> 4;
1893 regSize(sys_time) -> 5;
1894 regSize(tx_fctrl) -> 5;
1895 regSize(tx_buffer) -> 1024;
1896 regSize(dx_time) -> 5;
1897 regSize(rx_fwto) -> 2; % user manual gives 2 bytes and bits 16-31 are reserved
1898 regSize(sys_ctrl) -> 4;
1899 regSize(sys_mask) -> 4;
1900 regSize(sys_status) -> 5;
1901 regSize(rx_finfo) -> 4;
1902 regSize(rx_buffer) -> 1024;
1903 regSize(rx_fqual) -> 8;
1904 regSize(rx_ttcki) -> 4;
1905 regSize(rx_ttcko) -> 5;
1906 regSize(rx_time) -> 14;
1907 regSize(tx_time) -> 10;
1908 regSize(tx_antd) -> 2;
1909 regSize(sys_state) -> 4;
1910 regSize(ack_resp_t) -> 4;
1911 regSize(rx_sniff) -> 4;
1912 regSize(tx_power) -> 4;
1913 regSize(chan_ctrl) -> 4;
1914 regSize(usr_sfd) -> 41;
1915 regSize(agc_ctrl) -> 33;
1916 regSize(ext_sync) -> 12;
1917 regSize(acc_mem) -> 4064;
1918 regSize(gpio_ctrl) -> 44;
1919 regSize(drx_conf) -> 44; % user manual gives 44 bytes but sum of register length
    gives 45 bytes
1920 regSize(rf_conf) -> 58; % user manual gives 58 but sum of all its register gives
    53 => Placeholder for the remaining 8 bytes
1921 regSize(tx_cal) -> 13; % user manual gives 52 bytes but sum of all sub regs gives
    13 bytes
1922 regSize(fs_ctrl) -> 21;
1923 regSize(aon) -> 12;
1924 regSize(otp_if) -> 19; % user manual gives 18 bytes in regs table but sum of all
    sub regs is 19 bytes
1925 regSize(lde_ctrl) -> undefined; % No size ?
1926 regSize(lde_if) -> undefined; % No size ?
1927 regSize(dig_diag) -> 38; % user manual gives 41 bytes but sum of all sub regs
    gives 38 bytes
1928 regSize(pmsc) -> 44. % user manual gives 48 bytes but sum of all sub regs gives 41
    bytes
1929
1930 %% Gives the size in bytes
1931 subRegSize(agc_ctrl1) -> 2;
1932 subRegSize(agc_tune1) -> 2;
1933 subRegSize(agc_tune2) -> 4;

```

```

1934 subRegSize(agc_tune3) -> 2;
1935 subRegSize(agc_stat1) -> 3;
1936 subRegSize(ec_ctrl) -> 4;
1937 subRegSize(gpio_mode) -> 4;
1938 subRegSize(gpio_dir) -> 4;
1939 subRegSize(gpio_dout) -> 4;
1940 subRegSize(gpio_irqe) -> 4;
1941 subRegSize(gpio_isen) -> 4;
1942 subRegSize(gpio_imode) -> 4;
1943 subRegSize(gpio_ibes) -> 4;
1944 subRegSize(gpio_iclr) -> 4;
1945 subRegSize(gpio_idbe) -> 4;
1946 subRegSize(gpio_raw) -> 4;
1947 subRegSize(drx_tune0b) -> 2;
1948 subRegSize(drx_tune1a) -> 2;
1949 subRegSize(drx_tune1b) -> 2;
1950 subRegSize(drx_tune2) -> 4;
1951 subRegSize(drx_sfdtoc) -> 2;
1952 subRegSize(drx_pretoc) -> 2;
1953 subRegSize(drx_tune4h) -> 2;
1954 subRegSize(rf_conf) -> 4;
1955 subRegSize(rf_rxctrlh) -> 1;
1956 subRegSize(rf_txctrl) -> 4; % ! table in user manual gives 3 but details gives 4
1957 subRegSize(ldotune) -> 5;
1958 subRegSize(tc_sarc) -> 2;
1959 subRegSize(tc_pg_ctrl) -> 1;
1960 subRegSize(tc_pgdelay) -> 1;
1961 subRegSize(tc_pgtest) -> 1;
1962 subRegSize(fs_pllcfg) -> 4;
1963 subRegSize(fs_plltune) -> 1;
1964 subRegSize(fs_xtalt) -> 1;
1965 subRegSize(aon_wcfg) -> 2;
1966 subRegSize(aon_ctrl) -> 1;
1967 subRegSize(aon_rdat) -> 1;
1968 subRegSize(aon_addr) -> 1;
1969 subRegSize(aon_cfg0) -> 4;
1970 subRegSize(aon_cfg1) -> 2;
1971 subRegSize(otp_wdat) -> 4;
1972 subRegSize(otp_addr) -> 2;
1973 subRegSize(otp_ctrl) -> 2;
1974 subRegSize(otp_stat) -> 2;
1975 subRegSize(otp_rdat) -> 4;
1976 subRegSize(otp_srdat) -> 4;
1977 subRegSize(otp_sf) -> 1;
1978 subRegSize(lde_thresh) -> 2;
1979 subRegSize(lde_cfg1) -> 1;
1980 subRegSize(lde_ppindx) -> 2;
1981 subRegSize(lde_ppamp1) -> 2;
1982 subRegSize(lde_rxantd) -> 2;
1983 subRegSize(lde_cfg2) -> 2;
1984 subRegSize(lde_repc) -> 2;
1985 subRegSize(evc_ctrl) -> 4;
1986 subRegSize(diag_tm) -> 2;
1987 subRegSize(pmsc_ctrl10) -> 4;
1988 subRegSize(pmsc_ctrl11) -> 4;
1989 subRegSize(pmsc_snozt) -> 1;
1990 subRegSize(pmsc_txfseq) -> 2;
1991 subRegSize(pmsc_ledc) -> 4;
1992 subRegSize(_) -> error({error}).
1993
1994 %--- Debug -----
1995

```

```

1996 debug_read(Reg, Value) ->
1997     io:format("[PmodUWB] read [16#~2.16.0B - ~w] --> ~s -> ~s~n",
1998         [regFile(Reg), Reg, debug_bitstring(Value), debug_bitstring_hex(Value)]
1999     ).
2000
2001 debug_write(Reg, Value) ->
2002     io:format("[PmodUWB] write [16#~2.16.0B - ~w] --> ~s -> ~s~n",
2003         [regFile(Reg), Reg, debug_bitstring(Value), debug_bitstring_hex(Value)]
2004     ).
2005 debug_write(Reg, SubReg, Value) ->
2006     io:format("[PmodUWB] write [16#~2.16.0B - ~w - 16#~2.16.0B - ~w] --> ~s -> ~s~n",
2007         [regFile(Reg), Reg, subReg(SubReg), SubReg, debug_bitstring(Value),
2008             debug_bitstring_hex(Value)]
2009     ).
2010 debug_bitstring(Bitstring) ->
2011     lists:flatten([io_lib:format("2#~8.2.0B ", [X]) || <<X>> <= Bitstring]).
2012
2013 debug_bitstring_hex(Bitstring) ->
2014     lists:flatten([io_lib:format("16#~2.16.0B ", [X]) || <<X>> <= Bitstring]).

```

```

1 %% @doc This generic module defines the behaviour for any module implementing the
2 %% @end
3
4 -module(gen_mac_tx).
5
6 -export([start/2]).
7 -export([transmit/4]).
8 -export([stop/2]).
9
10 -include("pmod_uwb.hrl").
11 -include("ieee802154.hrl").
12 -include("ieee802154_pib.hrl").
13
14 %--- Callbacks -----
15
16 -callback init(PhyMod::module()) -> State::term().
17 -callback tx(State::term(),
18             Frame::bitstring(),
19             Pib :: pib_state(),
20             TxOptions::#tx_opts{}) -> {ok, Newstate::term()}
21                                     | {error,
22                                         Newstate::term(),
23                                         Error::tx_error()}.
24 -callback terminate(State::term(), Reason::atom()) -> ok.
25
26 %--- Types -----
27
28 -export_type([state/0]).
29
30 -opaque state() :: {Module::module(), Sub::term()}.
31
32 %--- API -----
33 -spec start(Module, PhyMod) -> State when
34     PhyMod :: module(),
35     Module  :: module(),
36     State  :: state().
37 start(Module, PhyMod) ->
38     {Module, Module:init(PhyMod)}.

```

```

39
40 -spec transmit(State, Frame, Pib, TxOptions) -> Result when
41     State      :: state(),
42     Frame      :: bitstring(),
43     Pib        :: pib_state(),
44     TxOptions  :: tx_opts(),
45     Result     :: {ok, State} | {error, State, Error},
46     Error      :: tx_error().
47 transmit({Module, Sub}, Frame, Pib, TxOptions) ->
48     case Module:tx(Sub, Frame, Pib, TxOptions) of
49         {ok, Sub2} ->
50             {ok, {Module, Sub2}};
51         {error, Sub2, Error} -> {error, {Module, Sub2}, Error}
52     end.
53
54 -spec stop(State, Reason) -> ok when
55     State      :: state(),
56     Reason     :: atom().
57 stop({Module, Sub}, Reason) ->
58     Module:terminate(Sub, Reason).

```

```

1 % @doc This module defines a generic behaviour for duty cycling on the IEEE
2   802.15.4
3 %
4 % The module implementing the behaviour will be responsible to manage the duty
5 %   cycling of the IEEE 802.15.4 stack (not the power optimization of the pmod)
6 % For example, the module implementing this behaviour for a beacon enabled network
7 %   will have the task to manage the CFP, the CAP and the beacon reception
8 % When an application will request a transmission the module has to suspend the rx
9 %   before transmitting
10 % At the transmission of a frame, the module will have the task to check if there
11 %   is enough time to transmit the frame (e.g. before the next beacon)
12 % At the transmission of a data frame with AR=1 the module has to manage the
13 %   retransmission of the frame if the ACK isn't correctly received
14 % This is because this module will be responsible to check if the retransmission
15 %   can be done (no beacons or not a CAP) and the reception can't be resumed
16 %   between retransmission (both are responsibilities of this module)
17 %
18 % Beacon enabled
19 % No transmission during beacon
20 % TX during CAP
21 % No TX during CFP unless a slot is attributed to the node
22 %
23 % Manage the RX loop (suspend/resume)
24 %
25 % @end
26 -module(gen_duty_cycle).
27
28 -include("ieee802154.hrl").
29 -include("ieee802154_pib.hrl").
30 -include("pmod_uwb.hrl").
31
32 -callback init(PhyModule) -> State when
33     PhyModule  :: module(),
34     State      :: term().
35 -callback on(State) -> Result when
36     State      :: term(),
37     Result     :: {ok, State}
38                 | {error, State, Error},
39     Error      :: atom().
40 -callback off(State) -> {ok, State} when

```

```

33     State :: term().
34 % Add suspend and resume later
35 -callback tx(State, Frame, Pib, Ranging) -> Result when
36     State      :: term(),
37     Frame      :: bitstring(),
38     Pib        :: pib_state(),
39     Ranging    :: ranging_tx(),
40     Result     :: {ok, State, RangingInfo}
41                 | {error, State, Error},
42     RangingInfo :: ranging_informations(),
43     Error       :: tx_error().
44 -callback terminate(State, Reason) -> ok when
45     State :: term(),
46     Reason :: term().
47
48 -export([start/2]).
49 -export([turn_on/1]).
50 -export([turn_off/1]).
51 -export([tx_request/4]).
52 -export([stop/2]).
53
54 %--- Types -----
55
56 -export_type([state/0, input_callback_raw_frame/0]).
57
58 -opaque state() :: {Module::module(), Sub::term()}.
59
60 -type input_callback_raw_frame() :: fun((Frame          :: binary(),
61                                         LQI            :: integer(),
62                                         UWBPRF         :: uwb_PRF(),
63                                         Security        :: ieee802154:
64                                             security(),
65                                         UWBpreambleRepetitions ::
66                                             uwb_preamble_symbol_repetition(),
67                                         DataRate         :: data_rate(),
68                                         Ranging          :: ieee802154:
69                                             ranging_informations())
69 -> ok).
70
71 %--- API -----
72
73 % @doc initialize the duty cycle module
74 % @end
75 -spec start(Module, PhyModule) -> State when
76     Module :: module(),
77     PhyModule :: module(),
78     State :: state().
79 start(Module, PhyModule) ->
80     {Module, Module:init(PhyModule)}.
81
82 % @doc turns on the continuous reception
83 % @TODO specify which RX module has to be used
84 -spec turn_on(State) -> Result when
85     State      :: state(),
86     Result     :: {ok, State} | {error, State, Error},
87     Error      :: atom().
88 turn_on({Mod, Sub}) ->
89     case Mod:on(Sub) of
90     {ok, Sub2} -> {ok, {Mod, Sub2}};
91     {error, Sub2, Error} -> {error, {Mod, Sub2}, Error}
92     end.

```

```

92 % @doc turns off the continuous reception
93 -spec turn_off(State) -> State when
94     State :: state().
95 turn_off({Mod, Sub}) ->
96     {ok, Sub2} = Mod:off(Sub),
97     {Mod, Sub2}.
98
99 % @doc request a transmission to the duty cycle
100 % The frame is an encoded MAC frame ready to be transmitted
101 % If the frame request an ACK, the retransmission is managed by the module
102 %
103 % Errors:
104 % <li> 'no_ack': No acknowledgment received after macMaxFrameRetries</li>
105 % <li> 'frame_too_long': The frame was too long for the CAP or GTS</li>
106 % <li> 'channel_access_failure': the CSMA-CA algorithm failed</li>
107 % @end
108 -spec tx_request(State, Frame, Pib, Ranging) -> Result when
109     State      :: state(),
110     Frame      :: bitstring(),
111     Pib        :: pib_state(),
112     Ranging    :: ranging_tx(),
113     State      :: state(),
114     Result     :: {ok, State, RangingInfo}
115                 | {error, State, Error},
116     RangingInfo :: ranging_informations(),
117     Error       :: tx_error().
118 tx_request({Mod, Sub}, Frame, Pib, Ranging) ->
119     case Mod:tx(Sub, Frame, Pib, Ranging) of
120     {ok, Sub2, RangingInfo} ->
121         {ok, {Mod, Sub2}, RangingInfo};
122     {error, Sub2, Err} ->
123         {error, {Mod, Sub2}, Err}
124     end.
125
126 % @doc stop the duty cycle module
127 -spec stop(State, Reason) -> ok when
128     State  :: state(),
129     Reason :: atom().
130 stop({Mod, Sub}, Reason) ->
131     Mod:terminate(Sub, Reason).

```

```

1 -module(duty_cycle_non_beacon).
2
3 -behaviour(gen_duty_cycle).
4
5 % gen_duty_cycle callbacks
6
7 -export([init/1]).
8 -export([on/1]).
9 -export([off/1]).
10 -export([tx/4]).
11 -export([terminate/2]).
12
13 % Include
14
15 -include("mac_frame.hrl").
16 -include("ieee802154_pib.hrl").
17 -include("ieee802154.hrl").
18
19 %% @doc
20 %% The module implementing this behaviour manages the duty cycling of the stack.

```

```

21 %% This includes:
22 %% <li>
23 %% IEEE 802.15.4 duty cycling:
24 %% Beacon enabled network, non-beacon enabled network
25 %% </li>
26 %% <li> pmod uwb duty cycling: low power listening, sniff mode </li>
27 %% @end
28
29 %--- Types -----
30
31
32 %--- Records -----
33 -export_type([state/0]).
34
35 -record(state,
36     {sniff_ont,
37      sniff_offt,
38      phy_layer,
39      loop_pid,
40      mac_tx_state}).
41 -opaque state() :: #state{}.
42
43 %--- gen_duty_cycle callbacks -----
44 -spec init(PhyMod) -> State when
45     PhyMod :: module(),
46     State  :: state().
47 init(PhyMod) ->
48     MacTXState = gen_mac_tx:start(unslotted_CSMA, PhyMod),
49     #state{sniff_ont = 3,
50            sniff_offt = 4,
51            phy_layer = PhyMod,
52            loop_pid = undefined,
53            mac_tx_state = MacTXState}.
54
55 -spec on(State) -> Result when
56     State  :: state(),
57     Result :: {ok, State} | {error, State, rx_already_on}.
58 on(#state{loop_pid = undefined} = State) ->
59     LoopPid = rx_loop_on(State),
60     {ok, State#state{loop_pid = LoopPid}};
61 on(State) ->
62     {error, State, rx_already_on}.
63
64 -spec off(State) -> {ok, State} when
65     State :: state().
66 off(#state{phy_layer = PhyMod, loop_pid = LoopPid} = State) ->
67     turn_off_rx_loop(PhyMod, LoopPid, shutdown),
68     {ok, State#state{loop_pid = undefined}}.
69
70 -spec tx(State, Frame, CdmaParams, Ranging) -> Result when
71     State      :: state(),
72     Frame      :: binary(),
73     CdmaParams :: pib_state(),
74     Ranging    :: ranging_tx(),
75     Result     :: {ok, State, RangingInfo}
76                 | {error, State, Error},
77     RangingInfo :: ranging_informations(),
78     Error       :: tx_error().
79 tx(#state{loop_pid = undefined} = State, Frame, CdmaParams, Ranging) ->
80     case tx_(State, Frame, CdmaParams, Ranging) of
81         {ok, NewMacTxState} ->
82             RangingInfo = tx_ranging_infos(Ranging, State),

```



```

83         {ok, State#state{mac_tx_state = NewMacTxState}, RangingInfo};
84         {error, NewMacTxState, Error} ->
85             {error, State#state{mac_tx_state = NewMacTxState}, Error}
86     end;
87 tx(State, Frame, CsmaParams, Ranging) ->
88     suspend_rx_loop(State),
89     TxStatus = tx_(State, Frame, CsmaParams, Ranging),
90     NewLoopPid = rx_loop_on(State),
91     case TxStatus of
92         {ok, NewMacTxState} ->
93             RangingInfo = tx_ranging_infos(Ranging, State),
94             {ok,
95              State#state{loop_pid = NewLoopPid,
96                          mac_tx_state = NewMacTxState},
97              RangingInfo};
98         {error, NewMacTxState, Error} ->
99             {error, State#state{loop_pid = NewLoopPid,
100                                mac_tx_state = NewMacTxState},
101              Error}
102     end.
103
104 -spec terminate(State, Reason) -> ok when
105     State :: state(),
106     Reason :: atom().
107 terminate(State, Reason) ->
108     LoopPid = State#state.loop_pid,
109     PhyMod = State#state.phy_layer,
110     MacTXState = State#state.mac_tx_state,
111     turn_off_rx_loop(PhyMod, LoopPid, Reason),
112     gen_mac_tx:stop(MacTXState, Reason),
113     ok.
114
115 %--- internal -----
116
117 %% @doc loop function for the reception
118 %% This function waits for a reception event to occur
119 %% If the event is the reception of a frame,
120 %% it will call the callback function to notify the next higher level/layer
121 %% If the event is an error, the function ignores it
122 %% @end
123 -spec rx_loop(PhyMod) -> no_return() when
124     PhyMod :: module().
125 rx_loop(PhyMod) ->
126     PhyMod:reception_async(),
127     rx_loop(PhyMod).
128
129 %% @doc
130 %% Sets the settings for reception and turns on the reception loop process
131 %% Returns the pid of the loop process
132 %% Note: this function will change when OS interrupts are introduced
133 %% @end
134 -spec rx_loop_on(State) -> pid() when
135     State :: state().
136 rx_loop_on(State) ->
137     PhyMod = State#state.phy_layer,
138     SniffOnT = State#state.sniff_ont,
139     SniffOffT = State#state.sniff_offt,
140     PhyMod:write(tx_fctrl, #{tr => 1}),
141     PhyMod:write(sys_cfg, #{rxwtoe => 0}),
142     PhyMod:write(pmsc, #{pmsc_ctrl1 => #{arx2init => 2#1}}),
143     PhyMod:write(rx_sniff, #{sniff_ont => SniffOnT, sniff_offt => SniffOffT}),
144     spawn_link(fun() -> rx_loop(PhyMod) end).

```

```

145
146 -spec turn_off_rx_loop(PhyMod, LoopPid, Reason) -> ok when
147     PhyMod  :: module(),
148     LoopPid :: pid() | undefined,
149     Reason  :: atom().
150 turn_off_rx_loop(_, undefined, _) -> ok;
151 turn_off_rx_loop(PhyMod, LoopPid, Reason) ->
152     PhyMod:disable_rx(),
153     unlink(LoopPid),
154     exit(LoopPid, Reason),
155     PhyMod:write(pmsc, #{pmsc_ctrl1 => #{arx2init => 2#0}}),
156     PhyMod:write(rx_sniff, #{sniff_ont => 2#0, sniff_offt => 2#0}),
157     PhyMod:disable_rx(),
158     PhyMod:write(sys_cfg, #{rxwtoe => 1}).
159
160 -spec suspend_rx_loop(State) -> ok when
161     State :: state().
162 suspend_rx_loop(State) ->
163     PhyMod = State#state.phy_layer,
164     LoopPid = State#state.loop_pid,
165     turn_off_rx_loop(PhyMod, LoopPid, shutdown).
166
167 % @private
168 -spec tx_State, Frame, Pib, Ranging) -> Result when
169     State  :: state(),
170     Frame  :: bitstring(),
171     Pib    :: pib_state(),
172     Ranging :: ranging_tx(),
173     Result :: {ok, MacTXState} | {error, MacTXState, Error},
174     Error  :: atom().
175 tx_State, <<_:2, ?ENABLED:1, _:13, Seqnum:8, _/binary>> = Frame, Pib, Ranging) ->
176     MacTXState = State#state.mac_tx_state,
177     PhyMod = State#state.phy_layer,
178     tx_ar(MacTXState, PhyMod, Frame, Seqnum, 0, Pib, Ranging);
179 tx_State, Frame, CsmaParams, Ranging) ->
180     MacTXState = State#state.mac_tx_state,
181     TxOpts = #tx_opts{ranging = Ranging},
182     gen_mac_tx:transmit(MacTXState, Frame, CsmaParams, TxOpts).
183
184 % @private
185 % @doc This function transmits a frame with AR=1
186 % If the ACK isn't received before the timeout, a retransmission is done
187 % If the frame has been transmitted MACMAXFRAMERETRIES times then the error
188 % 'no_ack' is returned
189 % @end
190 -spec tx_ar(MacTxState, PhyMod, Frame, Seqnum, Retry, Pib, Ranging) -> Result when
191     MacTxState :: gen_mac_tx:state(),
192     PhyMod     :: module(),
193     Frame      :: bitstring(),
194     Seqnum     :: non_neg_integer(),
195     Retry      :: non_neg_integer(),
196     Pib        :: pib_state(),
197     Ranging    :: ranging_tx(),
198     Result     :: {ok, MacTxState} | {error, MacTxState, Error},
199     Error      :: atom().
200 tx_ar(MacTxState, _, _, _, ?MACMAXFRAMERETRIES, _, _) ->
201     {error, MacTxState, no_ack};
202 tx_ar(MacTxState, PhyMod, Frame, Seqnum, Retry, Pib, Ranging) ->
203     TxOpts = #tx_opts{wait4resp = ?ENABLED, ranging = Ranging},
204     case gen_mac_tx:transmit(MacTxState, Frame, Pib, TxOpts) of
205     {ok, NewMacTxState} ->
206         case wait_for_ack(PhyMod, Seqnum) of

```

```

207         ok ->
208             {ok, NewMacTxState};
209         no_ack ->
210             tx_ar(NewMacTxState,
211                 PhyMod,
212                 Frame,
213                 Seqnum,
214                 Retry+1,
215                 Pib,
216                 Ranging)
217         end;
218     {error, NewMacTxState, _Error} ->
219         tx_ar(NewMacTxState,
220             PhyMod,
221             Frame,
222             Seqnum,
223             Retry+1,
224             Pib,
225             Ranging)
226     end.
227
228 wait_for_ack(PhyMod, Seqnum) ->
229     case PhyMod:reception(true) of
230     [_ , <<_:5, ?FTYPE_ACK:3, _:8/bitstring, Seqnum:8>>] ->
231         ok;
232     [_ , <<_:5, FType:3, _/bitstring>>] = Frame when FType /= ?FTYPE_ACK ->
233         ieee802154_events:rx_event(Frame, PhyMod:get_rx_metadata()),
234         no_ack;
235     _ ->
236         no_ack
237     end.
238
239 %--- Internal: Ranging helpers
240
241 tx_ranging_infos(?NON_RANGING, _) ->
242     #ranging_informations{ranging_received = false};
243 tx_ranging_infos(?ENABLED, State) ->
244     #state{phy_layer = PhyMod} = State,
245     #{rx_stamp := RxStamp} = PhyMod:read(rx_time),
246     #{tx_stamp := TxStamp} = PhyMod:read(tx_time),
247     #{rxtofs := RXTOFS} = PhyMod:read(rx_ttcko),
248     #{rxttcki := RXTTCKI} = PhyMod:read(rx_ttcki),
249     #ranging_informations{
250         ranging_received = true,
251         ranging_counter_start = TxStamp,
252         ranging_counter_stop = RxStamp,
253         ranging_tracking_interval = RXTTCKI,
254         ranging_offset = RXTOFS,
255         ranging_FOM = <<0:8>>
256     }.

```

```

1 -record(device, {slot, driver, pid, monitor}).

```

```

1 % This module has the responsibility of managing the channel access (CSMA/CA
   algorithm)
2 -module(unslotted_CSMA).
3
4 -include("ieee802154.hrl").
5 -include("ieee802154_pib.hrl").
6 -include("pmod_uwb.hrl").
7

```

```

8 -behaviour(gen_mac_tx).
9
10 -export([init/1]).
11 -export([tx/4]).
12 -export([terminate/2]).
13
14 %--- Macros -----
15
16 % According to Qorov forums, 1 symbol ~ 1 s => The unit of AUNITBACKOFFPERIOD
   are in s
17 -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.))
18
19
20 %% CCA Mode 5 should last at least the maximum packet duration + the maximum
   period for ACK
21 %% Maximum packet duration = 1207.79 s
22 %% Maximum period for ACK = 1058.21 s + 12 s
23 %% Sum => 2272 s
24 %% Since PRETOC units are in PAC size, we know that the default PAC is 8 symbols
   and 1 symbol ~ 1 s
25 %% We can conclude that CCA_DURATION = ceil(2272/8) = 284
26 % -define(CCA_DURATION, 284).
27
28 %--- Records -----
29
30 %--- gen_mac_tx Callbacks -----
31
32 -spec init(PhyMod) -> State when
33     PhyMod :: module(),
34     State  :: map().
35 init(PhyMod) ->
36     #{phy_layer => PhyMod}.
37
38 %% @doc Tries to transmit a frame using unslotted CSMA-CA
39 %% @param MacMinBE: The minimum value of the backoff exponent as described in the
   standard
40 %% @param MacMaxCSMABackoffs: The maximum amount of time the CSMA algorithm will
   backoff if the channel is busy
41 %% @param CW0: Not needed in this version of the algorithm. Ignored by this
   function
42 -spec tx(State, Frame, Pib, TxOpts) -> {ok, State} | {error, State,
   channel_access_failure} when
43     State  :: map(),
44     Frame  :: bitstring(),
45     Pib    :: pib_state(),
46     TxOpts :: tx_opts().
47 tx(#{phy_layer := PhyMod} = State, Frame, Pib, TxOpts) ->
48     CCADuration = math:ceil(cca_duration(PhyMod)),
49     PhyMod:write(sys_cfg, #{autoack => 0}),
50     MacMinBE = ieee802154_pib:get(Pib, mac_min_BE),
51     MacMaxBE = ieee802154_pib:get(Pib, mac_max_BE),
52     MacMaxCSMABackoffs = ieee802154_pib:get(Pib, mac_max_csma_backoffs),
53     PhyMod:set_frame_timeout(CCADuration),
54     Ret = case try_cca(PhyMod, 0, MacMinBE, MacMaxBE, MacMaxCSMABackoffs) of
55         ok ->
56             PhyMod:transmit(Frame, TxOpts),
57             {ok, State};
58         error ->
59             {error, State, channel_access_failure}
60     end,
61     PhyMod:write(sys_cfg, #{autoack => 1}),

```

```

62     Ret.
63
64 terminate(_State, _Reason) -> ok.
65
66 %--- Internal -----
67
68 % @doc Tries CCA until NB > maxCSMABackoff of if channel is detected idle
69 %
70 % The algorithm is described in figure 11 in sec. 5.1.1.4
71 %
72 % The timing settings to perform CCA shall be set prior to calling this func.
73 % @end
74 -spec try_cca(PhyMod, NB, BE, MacMaxBE, MacMaxCSMABackoffs) -> Result when
75     PhyMod      :: module(),
76     NB          :: non_neg_integer(),
77     BE          :: non_neg_integer(),
78     MacMaxBE    :: mac_max_BE(),
79     MacMaxCSMABackoffs :: mac_max_csma_backoff(),
80     Result      :: ok | error.
81 try_cca(_, NB, _, _, MacMaxCSMABackoffs) when NB > MacMaxCSMABackoffs ->
82     error;
83 try_cca(PhyMod, NB, BE, MacMaxBE, MacMaxCSMABackoffs) ->
84     PhyCfg = PhyMod:get_conf(),
85     RandBackOff = ieee802154_utils:symbols_to_usec(random_backoff(BE), PhyCfg),
86     SleepTime = trunc(math:ceil(RandBackOff/1000)),
87     timer:sleep(SleepTime),
88     case cca(PhyMod) of
89     ok ->
90         ok;
91     error ->
92         try_cca(PhyMod, NB+1, min(BE+1,MacMaxBE), MacMaxBE, MacMaxCSMABackoffs
93             )
94     end.
95
96 % @doc Performs CCA
97 -spec cca(PhyMod) -> Result when
98     PhyMod :: module(),
99     Result :: ok | error.
100 cca(PhyMod) ->
101     case PhyMod:reception() of
102     {error, rxrfto} -> ok;
103     {error, rxprd} -> error;
104     {error, rxsfdd} -> error; % theoretically, this should cover any frame rx
105     (i.e. channel is busy)
106     _ -> error % In case you receive a frame -> ? Could this happen ?
107     end.
108
109 % @doc Give the CCA duration in micro-seconds for mode 5
110 % According to sec. 8.2.7 the CCA period shall be no shorter than
111 % The maximum packet duration + maximum period for acknowledgment
112 %
113 % @end
114 cca_duration(PhyMod) ->
115     Conf = PhyMod:get_conf(),
116     TMaxPckt = ieee802154_utils:pckt_duration(127, Conf),
117     TAckPckt = ieee802154_utils:pckt_duration(5, Conf),
118     TurnAroundRxTx = 12, % us cf. datasheet sec. 5.1.6
119     ((TMaxPckt + TAckPckt) / ieee802154_utils:t_dsym(Conf)) + TurnAroundRxTx.
120
121 % @doc computes the backoff period (in symbol units)
122 % Table 11 - sec.5.1.1.4 says that this period shall be equal to:
123 % $$ \text{random}(2^{\text{BE}} - 1) $$ backoff units

```

```
122 % To get the value is symbol units => multiply the result by AUNITBACKOFFPERIOD
123 random_backoff(BE) ->
124     Backoff = round(math:pow(2, BE)) - 1, % [0, 2^BE-1]
125     rand:uniform(Backoff * ?AUNITBACKOFFPERIOD).
126     %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