Politecnico di Torino

Master's Degree in Electronic Engineering
A.a. 2023/2024
Sessione di Ottobre 2024

Development of a Monitor IP in a Power Line Communication Modem

Supervisors:
Prof. Guido MASERA

Candidate:
Riccardo GIUNTI

Company supervisors:
Ing. Cosimo ZACCARIA
Ing. Mauro BOSCO

Abstract

One of the greatest challenges of our time is the **decarbonisation**, for this reason the so called green mobility has gained an increasing priority on the european political plans throughout the recent years. This led the automotive industries to develop advanced **Electric Vehicles (EVs)** as well as dedicated **Electric Vehicle Supply Equipments (EVSEs)**. One of the main problems related to the Electric Mobility is charging, since it is an operation that must be controlled in such a way to be sustainable and optimized. For this reason it is important to establish a communication between the MCUs of EVs and EVSEs in order to make the Smart Grid to be aware of the demand of energy and so adjust it depending on the collected information on battery status and other features. For this purpose, the ADAS and Infotainment for Automotive team of **ST Microelectronics** developed its own **EV PLC Modem** whose target is to receive and transmit data in HomePlug Green PHY (HPGP) format on the power cable and exchange them with the host processor on the device. In particular, this specific data frame format has been adopted because it is supported by all the other similar devices that are actually in commerce since it grants the desired performances paying attention to the power consumption issue. The IP is composed mainly by an **HPGP subsystem** unit realized for ST by an external company which is in charge of modulate and demodulate the data to and from the HPGP format and by a Digital Kernel which implements communication, security and safety units as well as a microprocessor based on a RISC-V which controls all the system. An introductory part of this thesis is dedicated to the explanation of the overall EV PLC Modem architecture and to the main concepts regarding the HPGP standard that are behind the realization of the HPGP subsystem as well as the ones that are necessary to have a complete understanding of the **HPGP packet data frame**. Anyway, the main target of the thesis is to present the **Monitor IP**, which is the last digital unit that has been integrated in the Modem to increase its level of **functional safety**. This IP has the role to monitor some of the signals coming from the HPGP subsystem in order to recognize possible errors that may have been occurred in the transmitted frames or in part of them during the transmission because of noise or strong attenuation, to categorize them depending on the portion of the frame that is corrupted and to rise a fail signal if the percentages of these errors over the total number of received frames exceed some given thresholds that are set in registers via software. The aspect that this thesis wants to emphasize is the entire **workflow** that led to the realization of the Monitor IP. As a first thing, a set of corrupted frames has been obtained by introducing strong attenuation and high level noise on a starting set of correct frames by using Matlab, then these frames have been used in a simulation

environment to individuate some signals that could give information about the specific frame error. Given that signals, an initial RTL architecture of the Monitor has been implemented and then modified on the basis of the results of the synthesis on the FPGA used in the lab as a prototype. In the end, the results obtained in simulation have been confirmed by the comparison with the ones obtained from the lab test in FPGA. All of the functional steps involved in the Monitor realization can be considered as a complete example of integration of an IP in a more complex architecture.

Acknowledgements

Ringrazio il Prof. Masera per essermi stato di supporto durante la stesura di questa tesi e soprattutto per avermi trasmesso passione per le tematiche legate all'elettronica digitale affrontate durante le sue lezioni, fattore che ha contribuito principalmente alla scelta di questa tesi e alla conseguente posizione lavorativa acquisita.

Ci tengo inoltre a ringraziare in modo speciale i miei tutor di ST ed ora colleghi Mimmo e Mauro per avermi dato fiducia fin dal giorno zero, dandomi la grande opportunità di entrare in un' azienda leader del settore come ST Microelectronics nonché in un team molto affiatato come il loro, contraddistinto da un clima di amicizia e cooperazione che mi ha permesso di integrarmi da subito nel migliore dei modi, consentendomi di poter esprimere me stesso al meglio sia da un punto di vista lavorativo che umano.

Un grande grazie va anche ai miei amici, con i quali ho un rapporto sincero e di lunghissima data che sono sicuro continuerà a durare invariato nel tempo nonostante la vita ci presenterà sempre più impegni. Mi reputo molto fortunato ad avervi avuto accanto in un periodo bellissimo e spensierato ma allo stesso tempo delicato come l'adolescenza. Nonostante siate diversi tra voi, penso sinceramente che ognuno mi abbia lasciato qualcosa, contribuendo positivamente allo sviluppo della mia persona.

Per ringraziare i miei genitori invece non basterebbe il centinaio di pagine di questa tesi. Mi avete sempre sostenuto a tutto tondo in ogni mia scelta, senza mai una volta farmi pesare eventuali sacrifici né tantomeno mettendomi pressioni sui risultati come altri genitori fanno. Se oggi mi sento una persona felice lo devo in grandissima parte al vostro modo di essere e di avermi cresciuto. Per il futuro spero di continuare a rendervi orgogliosi e di restituire, anche solo in parte, tutto quello che mi avete dato.

Mi è doveroso inoltre ringraziare tutto il resto della mia famiglia per avermi sempre fatto sentire apprezzato e sostenuto per ogni mia decisione, contribuendo a

costruire intorno a me un ambiente ideale in cui crescere e maturare. Ringrazio in modo particolare mio zio Enrico per avermi trasmesso il suo amore per la chitarra, hobby che mi ha alleggerito le giornate di studio e anche mia nonna Lia ed Egidio che oltre al loro affetto e ammirazione mi hanno fornito più volte un supporto pratico per i miei spostamenti verso Torino. Vorrei dedicare parte di questa tesi anche a mio nonno Francesco, figura alla quale ho sempre guardato con rispetto e ammirazione e che sono sicuro avrebbe gioito per me in questo momento.

Per ultima, non certo per importanza, voglio ringraziare Isi che da sette anni a questa parte è la persona che mi completa e con la quale ho condiviso le esperienze più belle. Quando ci siamo conosciuti eravamo poco più che maggiorenni e durante questo tempo abbiamo imparato a conoscerci, capirci, sostenerci e ad imparare l'uno dall'altra. In questi anni sei stata la persona per la quale ho sempre cercato ogni giorno di tirare fuori la parte migliore di me e oggi posso dire che ne è sempre valsa la pena e non ho intenzione di smettere di farlo perché penso che la nostra relazione sia speciale e destinata a farci condividere ancora molti momenti indimenticabili.

Dedico quindi questa tesi a tutti voi che, direttamente o indirettamente, avete contribuito alla sua realizzazione.

Table of Contents

List of Tables

List of Figures

Acronyms

AFE

Analog Front-End

AGC

Automatic Gain Controller

APB

Advanced Peripheral Bus

AVLN

HomePlug AV Logical Network

BMS

Battery Management System

BPCnt

Beacon Period Count

BTO

Beacon Transmission Offset

BTS

Beacon Time Stamp

CCo

Central Coordinator

CRC

Cyclic Redundancy Check

CSMA

Carrier Sense Multiple Access

DMA

Direct Memory Access

EU

European Union

EV

Electric Vehicle

EVSE

Electric Vehicle Supply Equipment

FEC

Forward Error Correction

FFT

Fast Fourier Transform

GV

Greenvity

HDL

Hardware Description Language

HPGP

HomePlug Green PHY

HPAV

HomePlug AV

IFFT

Inverse Fast Fourier Transform

IP

Intellectual Property

LPF

Low Pass Filter

MAC

Media Access Control Layer

MDIO

Management Data Input/Output

NTB

Network Time Base Synchronization

OBC

On Board Charger

ODA

Original Destination Address

OFDM

Orthogonal Frequency Division Multiplexing

OSA

Original Source Address

PB

PHY block

PBB

PHY Block Body

PBCS

PHY Block Check Sequence

PEV

Plug-In Electric Vehicle

PHY

Physical Layer

PLC

Power Line Communication

PLL

Phase Locked Loop

PMU

Power Management Unit

ROBO

Robust OFDM

RSC

Recursive Systematic Convolutional

RSSI

Received Signal Strength Indicator

SACK

Sound Acknowledge

SLAC

Signal Level Attenuation Characterization

SNR

Signal to Noise Ratio

SOF

Start of Frame

SSN

Segment Sequence Number

STA

AV network station

TEI

Terminal Equipment Identifier

Chapter 1

Introduction

1.1 Social and political context of the thesis

Nowadays one of the most discussed policies of the **EU** is the **green mobility**. As a matter of facts as stated by the **European Council**, the **Decarbonisation** must be one of the main targets of the Union, especially the one related to the **transport sector**.

Under the **Paris Agreement**, the nations under the EU are required to become climate-neutral by 2050. Of course it is a very ambitious deadline that maybe will be slightly extended in order to give more time to the european countries to adapt to this new condition also in a more economically sustainable way, however it becomes crucial to start making changes, especially from the means of transport that represent one quarter of the **EU greenhouse gas emissions**. The following picture 1.1 taken from the official website of the **European Council** [1] gives an idea of what has been just stated. As it can be seen from 1.1, the transport sector is a tough obstacle to overcome for reaching the complete decarbonisation, its emissions should be decreased of the 90% with respect to the levels of the 1990 by 2050.

1.2 The problem of charging

For all the reasons above, it becomes of fundamental importance to concentrate most of the efforts in producing and improving **Electric Vehicles (EVs)** . In particular one the main challenges regarding the theme of the EVs is their **charging**, so the optimized interaction between them and their **Electric Vehicle Supply Equipments (EVSEs)**. This because of course it is important aspiring to be

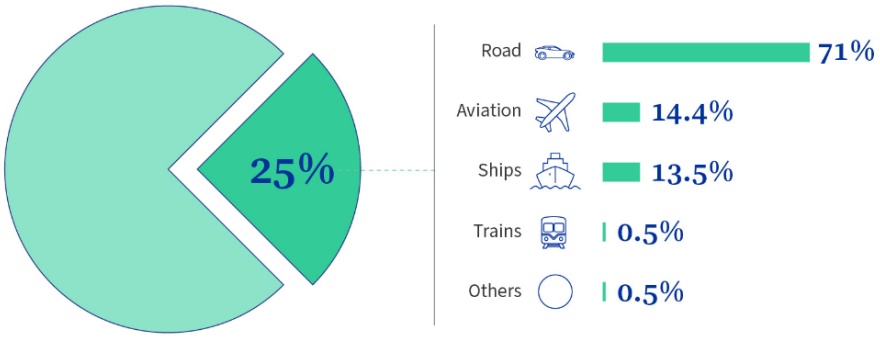

Figure 1.1: Greenhouse gas emissions due to means of transport in EU (taken from [1])

carbon-neutral but at the same time also the use of the electricity must be sustainable and optimized.

This is directly connected with the development of **Smart Grid Technology**. A **Smart Grid** is an energy network that is able to monitor the flow of energy and adjust it depending on the demand [2]. In a practical use case correlated with the topics of this thesis, there can be the possibility that more than one EV are connected to the same grid (by mean of their EVSEs) that eventually has to split the power that it provides. In this case it is important that the local grid can share information with the vehicles in order to optimize the charging operation for every EV otherwise for granting a good charging experience to users, the local grid would have to provide at the same time to all EVs a huge amount of power and that would be incredibly costly.
This means that the local grid, through the information that can get from the vehicles by mean of the EVSE, can be aware of the **load** (so basically the number of vehicles connected) and other specifics as the **speed of charging** and the **maximum charging capacity** of each vehicle connected in order to supply power in a smart way.

Due to this need of exchanging information between the EV and the electrical grid that delivers electricity (now is for that reason that is defined a "smart" grid) in an optimized way, several **Power line communication (PLC)** standards have been emerged. In a PLC, the power grid is intended as a channel not only for conveying power but also data messages and it can be convenient to adopt because of its cost-effectiveness and deployment without the need of additional cables [3].

However through the use of a PLC, the information is carried by the power cable in a specific format that is dependent from the specific PLC standard that has been adopted. This leads to the necessity of implementing a **modem** that is useful for modulating the digital information in the PLC frame format and at the receiver side for demodulating it in order to recover the digital information. The need for this kind of device made ST Microelectronics to develop its own **EV PLC modem** for placing itself in this emerging market.

1.3 Thesis purposes and structure

Given the context that has been presented in the last two sections, the first purpose of this thesis is to present and explain the main concepts regarding the **Home Plug Green PHY** that is the PLC communication standard managed by the **EV PLC modem** that has to receive and transmit HPGP data and exchange them with an host processor. All the information that are contained in chapter 2 do not represent an exhaustive summary of the content of the **HPGP standard** [4] since it is a very complex and verbose documentation but a presentation of the basic theoretic contents that are necessary to understand the **EV PLC modem** project and its **HPGP monitor**.

After the explanation of the communication standard, in chapter 3 the focus is shifted towards the **Electric Vehicle Power Line Communication modem (EV PLC modem)** project developed with the contribution of the **MDRF RFOC Digital Audio and Signal Solutions Division** team of ST Microelectronics based in Cornaredo (MI) which is the IP that represents the environment in which the Monitor IP, that is the main character of the thesis, is inserted. This IP has been designed to manage the Power Line communication between the EV and the EVSE. In particular Chapter 3 reports its main aspects and characteristics taken from the official documentation of the project and some commented block diagrams that can help the reader to understand the architecture of the developed model from top to down level of abstraction.

The **HPGP monitor** is a component of the EV PLC modem and represents my personal contribution in the team. It consists in the realization of a functional monitor that has been inserted in the project which is useful to check the correctness of the received data frame and eventually to signal every sort of malfunction. A monitoring IP in the automotive field represents an important feature because if something goes wrong, a robust and reliable system must be aware of that before the malfunction can cause a significant damage to the entire system and eventually the user. This topic has been introduced in chapter 4 where all the phases of its

development have been exposed, starting from the study on the possible errors that can corrupt a frame for going through the set up of a proper simulation environment and its architecture realization in Hardware Description Language (HDL), ending with the simulation of the latter and its validation on a FPGA.

Chapter 2

HomePlug Green PHY

In this chapter are contained all the relevant information of the **HPGP standard** which is the power line communication protocol that have been used in order to understand the theory underlying the project and its development. The **HPGP standard** [4] is a specification document released in 2012 by **HomePlug Powerline Alliance** that is a trade association of electronics manufacturers that establishes standards for the several **power line communication technologies** at the basis of their products. The document can be seen as a guide to understand this kind of technology for design and use purposes along with **HomePlug AV and IEEE 1901: a handbook for PLC Designers and Uses** [5] that has been used as a parallel guide for the HPGP comprehension since it describes in details the **HomePlug AV** which is the standard from which the HPGP architecture is derived.

2.1 Why using the Home Plug Green PHY

The basic principle of a Power Line Communication is exchanging information between digital devices through the use of the power lines. This kind of technology is widely used in many applications because it allows the communication between devices that are inserted in the same "power environment" without the need of making a new wiring. Probably the most known case is the **home internet connection** as presented in the scheme reported in figure 2.1 taken from the official website of the **HomePlug Powerline Alliance** [6]. As it can be seen, the network connection coming from the Router is sent through the home power lines by mean of two units that act respectively as transmitter and receiver where the first modulate the digital information from the Router in order to transmit it in the power line while the latter demodulate the received frame in order to recover the digital information for the PC.

Figure 2.1: Network connection through a PLC (taken from [6])

To establish a **Power Line Communication**, many protocols have been developed and categorized by the HomePlug Alliance during the years depending on the kind of application and of the performances, however the most successful of these standards is the **HomePlug AV** and it has been the result of several years of research to implement a protocol that could be able to optimize the performances for the power line environment. It has been realized because of the need to find a new LAN technology for transporting the Internet connection without the need to rely on additional cables. The creation of this protocol is strongly correlated with the establishment of the **IEEE 1901** standard for high-speed PLC systems which is mandatory in many fields of application such as the **EV DC charging** which is the one of main importance in this thesis. As a matter of facts it represents a sort of compromise between the **HomePlug AV** and the standard used in **Panasonic's HD-PLC devices**. Due to this fact, many existent devices have been designed taking into account a compatibility with this standard.

Despite the performances offered by the HPAV are very high, in EV charging field the focus is more shifted towards the **reliability** and especially the **energy saving**. This need led HomePlug Alliance to think about a more power saving alternative to the HPAV that is the **HomePlug Green PHY** which permits to transport information with strong reliability using very little energy at the cost of reduced performances.

2.2 HomePlug AV and HomePlug Green PHY main features

As introduced previously, the two protocols are strongly correlated since the HPGP is derived from the HPAV also because many of the existent devices make use of the HPAV and so the HPGP has been developed taking into account that it would be interoperable with the HPAV protocol in a possible heterogeneous network.

However there are some differences in the specifications that are reported in the following table 2.1 derived from the information taken from the **HPGP whitepaper** [7] which is a reduced version of the standard released by the Power Alliance.

Parameter	**HomePlug AV**	**HomePlug GP**
Spectrum	2 MHz to 30 MHz	2 MHz to 30 MHz
Modulation	OFDM	OFDM
# Subcarriers	1155	1155
Subcarrier spacing	24.414 kHz	24.414 kHz
Supported subcarrier modulation formats	BPSK, QPSK, 16 QAM, 256 QAM, 1024 QAM	QPSK only
Data FEC	Turbo code Rate 1/2 or Rate 16/21 (punctured)	Turbo code Rate 1/2 only
Supported Data rates	ROBO: 4 Mbps to 10 Mbps Adaptive Bit Loading: 20 Mbps to 200 Mbps	ROBO: 4 Mbps to 10 Mbps

Table 2.1: HomePlug GP PHY Simplifications Reduce Cost & Power Consumption

First of all it must be said that the differences that can be found out by looking at the table regard the **Physical layer** of the HomePlug protocols. As a matter of facts both the protocols' architectures (as most of the other HomePlug devices) make use of two main entities: the already mentioned **Physical layer (PHY) and the Media Access Control layer (MAC)** where the first one is encharged of physically modulate the signal into HPGP frames while the latter transfers the digital data from microprocessor and so the digital kernel of the device to the PHY and viceversa. So in practice between the two protocols there are differences in how the digital information is modulated in order to be transferred as an HPGP frame through the power cable.

Concentrating first on the similarities between HPAV and HPGP, by looking at table 2.1 it can be seen that the signals have the same bandwidth from 2MHz to 30MHz. This means that for most of other electronics devices that do not use these standards these signals are seen as **high frequency noise**. This is important because this peculiarity make these signals to not disturb other appliances.

In addition both the PHY layers adopt the **Orthogonal Frequency Division Multiplexing (OFDM)** modulation technique. Since it is a **Frequency Division Multiplexing** technique, the main channel at a certain frequency band is

splitted in multiples subchannels that are at different subcarriers. In the case of the **Orthogonal Frequency Division Multiplexing**, the bitstream of information is subdivided in orthogonal and parallel subcarriers. This feature is important because it allows to reduce the **distructive interference** coming from the **Multipath distortion** that is an effect that happens when a signal recombines after having reached a receiver device multiple times through multiple paths. As it can be seen from the table the **number of subcarriers** as well their spacing is the same in the two protocols.
One of the key advantages of this kind of modulation is of course the **signal integrity**. As a matter of fact, splitting the signal across a certain number of subchannels instead of spreading it on a single wideband channel allows to represent each bit on a bigger time interval and also to increase the time spacing between adjacent bits. In this way the data rate is the same but the signal integrity is higher, and this is crucial for the project exposed in this thesis since in a Smart Grid with more EVSEs, high amplitude noise can be easily introduced.

Another common feature is the **Data Forward Error Correction (FEC)** that is performed by mean of **Turbo codes**. The FEC is a digital signal processing feature that consists in the implementation of an algorithm or a code that adds some **parity bits** to the transmitted data that than will be decoded on the receiver side and used to recognize if an error occurred during the communication and eventually correct it. There are several methods to implement the FEC, in the case of both the HPAV and the HPGP, **Turbo codes** have been used.
According to the document [8], Turbo Codes represented a revolution in the digital communications. This because before their invention, the communication engineers thought that in order to reduce communication errors, the only possible way was to increase significantly the transmission power or as an alternative sending the message repeatedly multiple times on the channel. Since that was not a nice power efficient way to make communications, some researchers as **Shannon** demonstrated that theoretically making use of some **error-correction codes** it was possible to transmit a free of error information nearly at the maximum channel capacity and using little energy. After this theoretical discovery many researchers developed some good error-correction codes but still using more power than Shannon's research stated it was sufficient.
Then **A. Berrou** and **A. Glavieux** invented Turbo Codes that allow to get about 0.5 dB of the maximum channel capacity for a BER of 1/100000 using a very little energy.
The concept behind the idea of Turbo code is "divide and conquer". As a matter of facts the standard methods for implementing error-correction codes involved an encoder that has to generate **parity bits** to be added to the original digital information at the transmission side and a decoder at the receiving one for observing

eventual changes in the received parity bits. This led to an incredible computational complexity when dealing with a lot of **parity bits** needed for not only detecting an error but also correcting it. In a Turbo code instead, two encoders and decoders are used in parallel, in order to split the computational cost and increase the reliability. The degrees of freedom of the method are the type of encoders, input/output ratios and kind of **puncturing patterns** that are blocks that are used in practical Turbo code implementations (such as the one of the HPGP) for removing some of the parity bits after the encoding in order to obtain an higher code rate (less redundancy) increasing the flexibility of the system.
In the case of the HPGP are used two **RSC encoders**. However the architectural implementation of the Turbo code used in HPGP will be treated in details in the following sections.

Coming back to the comparison between the HPAV and the HPGP, one of the main differences between the two protocols is the exclusive use of the **QPSK subcarrier modulation** of the HPGP with respect to the HPAV that makes use of all the modulations reported in table 2.1.
As a matter of facts, the HPGP does not allow the **Adaptive Bit Loading** mode. This because the latter allows to assign to every subcarrier the proper **modulation format**: if a subcarrier is strongly attenuated it will carry for instance just 2 bits of data and so a **BPSK** modulation will be the best one while if it is less attenuated it can be assigned for example to a **1024 QAM** that gives a bit depth of 10 bits. From the moment that the HPGP supports for every subcarrier just the QPSK modulation the Adaptive Bit Loading is of course useless. Adaptive Bit Loading is used by the HPAV in order to achieve the highest possible data rate given the signal strength at the receiver side so it enables every subcarrier to carry as much data as possible for the given line conditions. In order that this can be possible the transmitter must be able to know the attenuation of each subcarrier at the receiver. This is done by the exchanging of **sounding packets** between the tx and rx devices in the HPAV network. The information in the sounding packets will compose the so called **Tone maps** that keep track of the attenuation of the subcarriers of all the receivers in the HPAV network. This system introduces complexity but at the same time allows to reach a peak PHY rate of Mbps. Since in the HPGP the two main goals are reliability and reduced power consumption, it was decided that in its case the **peak data rate** could be reduced since it is not necessary to have such high performances for Smart Grid applications and moreover this in turn reflected to a reduction in costs and consumptions.

A last aspect that must be considered in this treatment is the adoption of **ROBO modes** for both HPAV and HPGP. ROBO stands for **Robust OFDM** and it means that the bitstream of information is sent in a redundant way on multiple

carriers in order to increase the reliability during the transmission. There are 3 ROBO modes supported by the HPGP: **Mini-Robo**, **Standard Robo** and **High Speed Robo**. Of course if the number of repetitions of bitstream per subcarrier is higher, the reliability is increased but the throughput is reduced and viceversa. The following table 2.2 reports the just mentioned modes.

Mode	PHY Rate	# Repeat Copies
Mini-ROBO	3.8 Mbps	5
Standard ROBO	4.9 Mbps	4
High Speed ROBO	9.8 Mbps	2

Table 2.2: ROBO modes with related Data Rate and Number of Copies

2.3 Power Save Mode for Green PHY

It has been reported multiple times that **power saving** is crucial for Smart Grid applications. The reason why HomePlug Green PHY is strongly suggested for this application field, is because it is characterized by a special **Power Save Mode**. In this mode, the device is able to reduce its **average power** by alternating awake and sleep states. In general, there can be distinguished four different periods of time in which devices that make use of HPGP can operate:

1. **Awake window**: time range from 1.5 ms to 2.1 ms in which the device can transmit or receive frames on the power line.
2. **Sleep window**: the device is not able to transmit or receive frames over the power line.
3. **Power Save Period (PSP)**: it is the sum of awake window duration and the sleep one.
4. **Power Save Schedule (PSS)**: it conveys the PSP and the Awake window duration. It is essential because a station (device) that is operating in power save mode has to communicate the duration of its operating windows to the other devices in the network and so does it by communicating its PSS timing.

As it will explained in the next sessions, in the most of the applications one can refer to a bunch of devices that use HPGP or HPAV to a **HomePlug AV Local Network (AVLN)** that is coordinated by one of these devices and it takes the name of **Central Coordinator (CCo)** while the other devices are named

Stations (STAs). Among the other actions that has to perform, the CCo must coordinate the power state mode for all the STAs in the AVLN including itself. As it is well explained in [5], before entering the Power Save Mode a STA sends to the CCo a message that contains its PSS and so it responds to the STA with another message that basically communicates if the request to enter the Power Save Mode has been accepted. After that the CCo provides a signal named **Beacon Period Count (BPCnt)** that as the name suggests represents a signal counting all the beacons the CCo has delivered until that moment. Also the concept of Beacon will be analyzed later, however it is a sort of message that the CCo sends in the network in order to synchronize all the other STAs. Also in this case the purpose is the synchronization, as a matter of facts that BPCnt signal is used to enable STAs to synchronize the start of their Awake window. In the following figure 2.2 there is an example of Power Save scheduling synchronization in a network taken from [7].

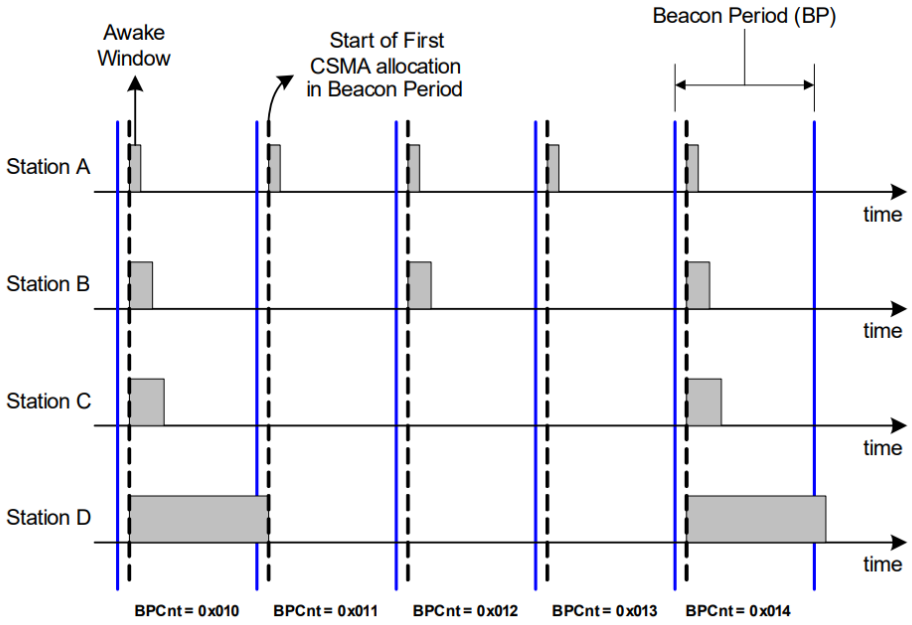

Figure 2.2: Examples of Power Save Schedules (taken from [7])

2.4 Plug-in Vehicle (PEV) Association

One of the big advantages of making use of a PLC like the HomePlug Green PHY is the possibility to provide a smart charging experience to users potentially in every possible location. On the other hand this feature represents also a potential risk since a **PEV charging station** can be installed in areas that lack security

and many private data such as the **billing information** of the user may be compromised.

Looking at the following figure 2.3 taken from [7], one can notice that one problem related to the security can be represented by the **association** of the PEV to a specific EVSE of the Smart Grid. This because it is common that PEVs and EVSEs

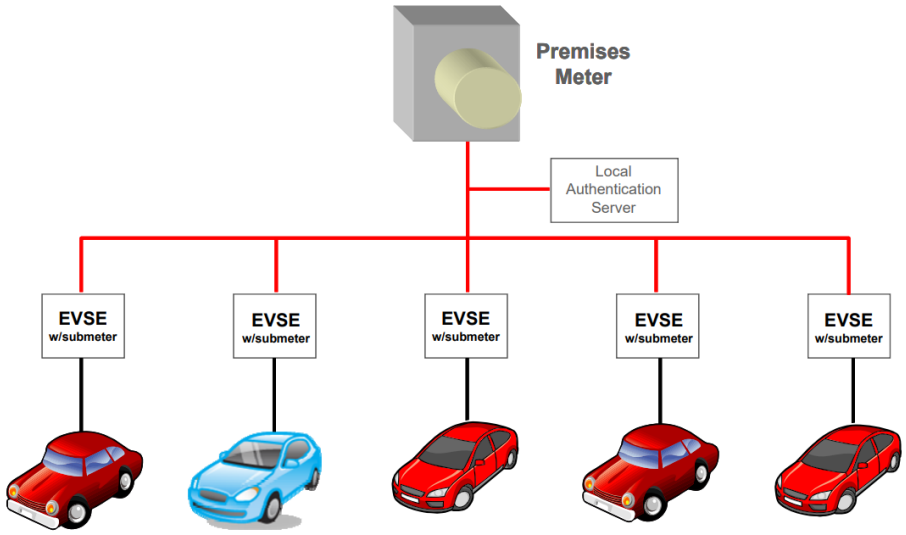

Figure 2.3: PEV/EVSE Association (taken from [7])

are located in close proximity since EVSEs are installed in a space limited Smart Grid area. This leads to the possibility that some signals transmitted by a PEV to the associated EVSE may be sensed also by other nearly located EVSEs. Since the other key feature of the HPGP must be **reliability**, to overcome this possible issue the **Signal Level Attenuation Characterization (SLAC)** has been added to the specifications.
When the SLAC is used, the EV sends a **sounding packet** to each of the EVSEs connected in the smart grid and at their turn the EVSEs calculate the signal power of the signal they received and send it back to the EV with also their **MAC address**. In this way the EV can connects to the MAC address of the EVSE that sensed the highest signal power so that the connection is established with the nearest EVSE.
This kind of connection is safe because it assures that control signals are sent through PLC to the EVSE to which the vehicle is plugged. In this way, the **charging contact** between the EV and the local infrastructure in closed so that the charging operation can start, at the same time an open switch and a **signal bypass capacitor** attenuate heavily the PLC signal that can propagates past the

EVSE and maybe interferes with the charging of other vehicles in the smart grid. So by measuring and reporting the received PLC signal strength among all stations receiving the **sounding packets**, the **SLAC** procedure reliably facilitates **PEV/EVSE association**. The above situation is represented in the following figure 2.4:

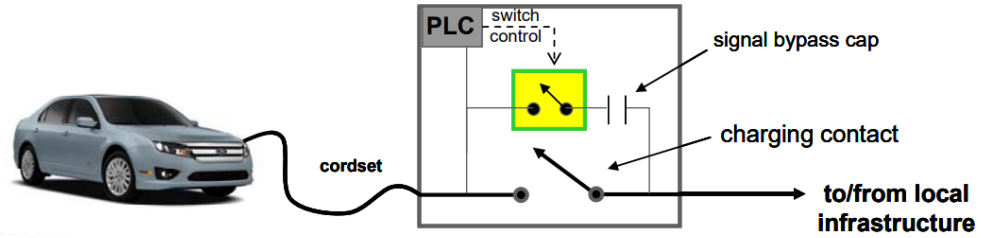

Figure 2.4: Bypass capacitor permits signals traverse open charging contacts (taken from [7])

2.5 HomePlug Green PHY system overview

In this section it will be given a generic overview of the system that supports the HomePlug Green PHY standard of communication taking into consideration that a more detailed analysis of the HPGP subsystem architecture and its composing blocks will be given in chapter 3. In order to start from the highest possible abstraction level, the functional block diagram exposed in figure 2.5 can be analyzed.

As it can be seen, from the most abstract point of view, the system architecture related to the HPGP communication standard is composed mainly by three layers whose definitions has been given in [4]:

- **PHY layer** which performs forward error correction (FEC) and maps data onto OFDM symbols generating requisite time domain waveforms.
- **MAC layer** which determines the correct position of transmission and formats the data frames into fixed-length entities for transmission on the channel.
- **Convergence layer** that performs bridging, classification of traffic into connections, and data delivery smoothing functions.

Note that these are generic and introductory definitions that will be better explained in the following specific sections on the architecture. Moreover they are related to the transmission, so the receive side performs the same operations but in reverse.

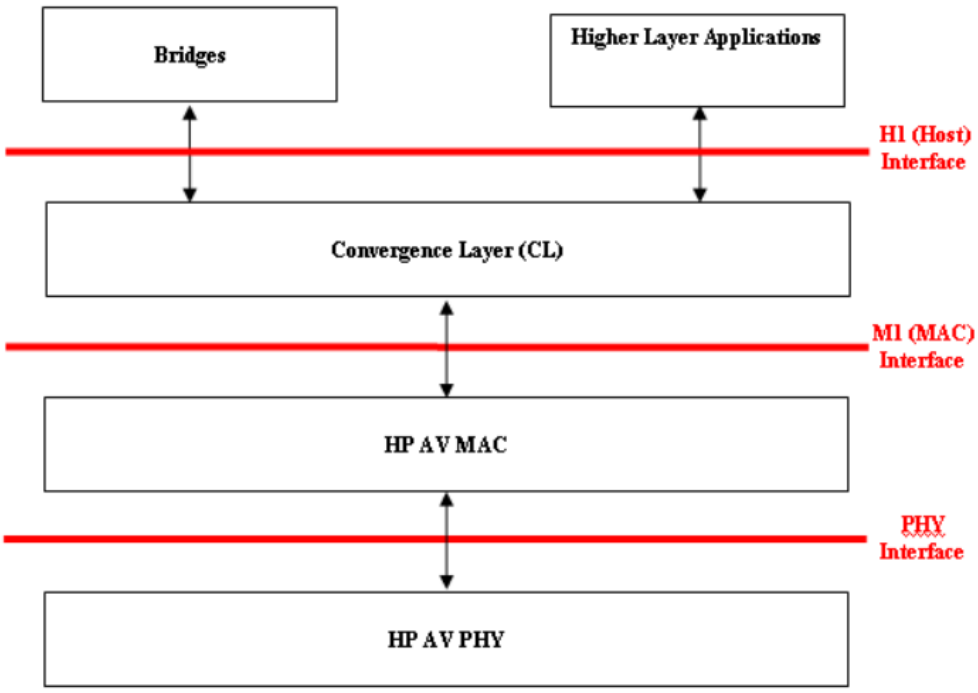

Figure 2.5: System reference model (taken from [5])

In particular the PHY layer as its name suggests represents the unit that **physically** converts the data in the HPGP format, performing the modulation/demodulation of the data in order to obtain OFDM symbols to be exchanged on the power line or recover from them digital data. The function of the MAC instead, is to transfer data from an host microprocessor architecture to the PHY with or without encryption on TX side, while on the RX side the operation is reversed. So in a certain sense, the MAC acts as a bridge between the data coming from the microprocessor and the PHY, which transforms these data in a format that is suitable for the communication. As for the Convergence Layer, it is an entity which interfaces with the **host devices** and exchange information with them, eventually formatting the data and delivering them to the appropriate destination. However for the purposes of this thesis the attention will be payed for the **PHY** and **MAC** layers which are the **top entities** that are directly responsible of manipulating the communication frame.

At this point of the architecture overview, it is important to deal with the concept of **Network Architecture** since a HomePlug Green PHY powerline network consists in a set of **HomePlug stations** connected to the AC powerline. For this purpose, the following figure 2.6 can be considered.

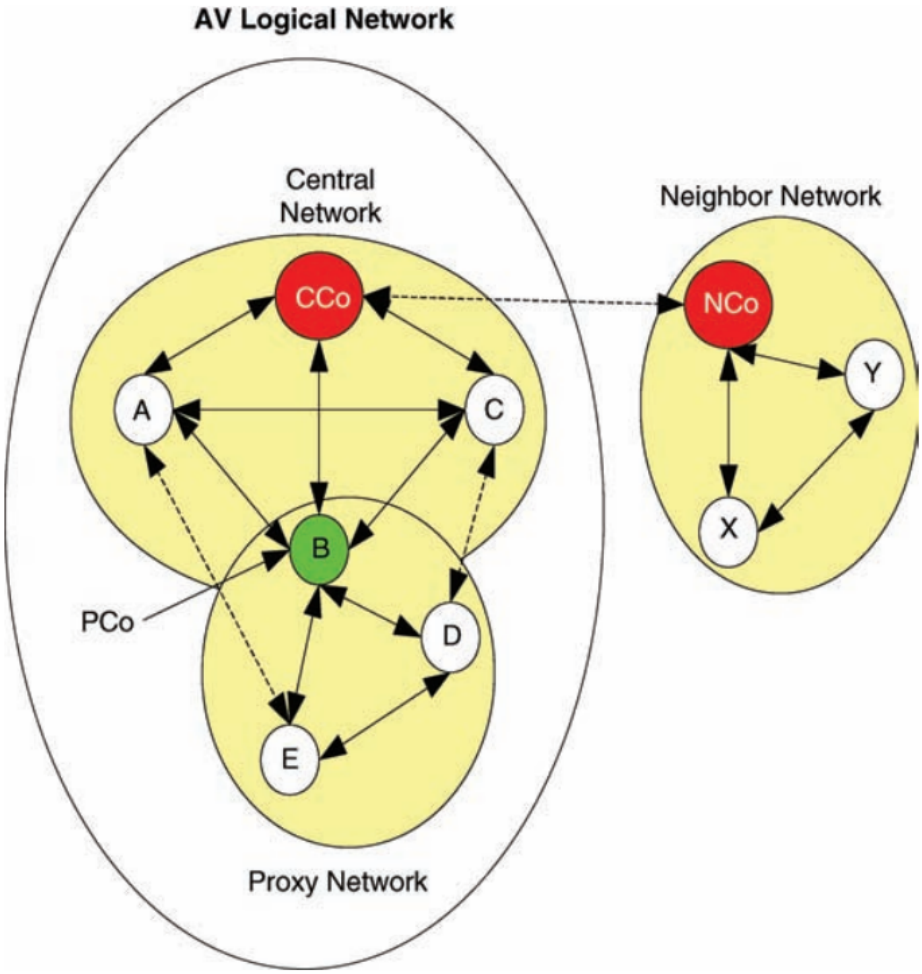

Figure 2.6: HomePlug Green PHY network architecture (taken from [5])

The **AV Logical Network (AVLN)** is the set of **HomePlug powerline stations (STAs)**, typically used in a home environment, that possess the same **Network Identifier (NID)** and **Network Membership Key (NMK)**. The **CCo** performs network management functions such as authentication and association of new stations joining the AVLN, AC line cycle synchronization of transmission intervals, and admission control and scheduling for **Carrier Sense Multiple Access (CSMA)** sessions and allocations. During the association process, the CCo provides the new station with a **Terminal Equipment Identifier (TEI)** which is used to identify a station uniquely within the AVLN. Both the choice of the CCo and a detailed explanation of the CSMA will be given in chapter 3 when dealing with the description of the MAC layer architecture.

In the specific case of the **EV PLC modem** project the concepts that are important to explain are those which concern the AVLN, the STA and the CCo that form the so called **Central Network** that so in the mentioned field of application coincides with the AVLN itself. In figure 2.6 can be distinguished also the **Proxy Network** and the **Neighbor Network** where the first one represents the network of the hidden stations for the CCo and the second one is like a parallel network to the one of the CCo of the central network, however these two concepts have been not taken so much in consideration for the development of the project so they will not be explained in details.

2.6 HPGP packet data frame

In order to explain the formation and composition of the data frame in the context of the HomePlug Green PHY communication, it can be assumed to have a station that from the Higher Level Entity (HLE) receives some data in **Ethernet format** as depicted in the following figure 2.7.

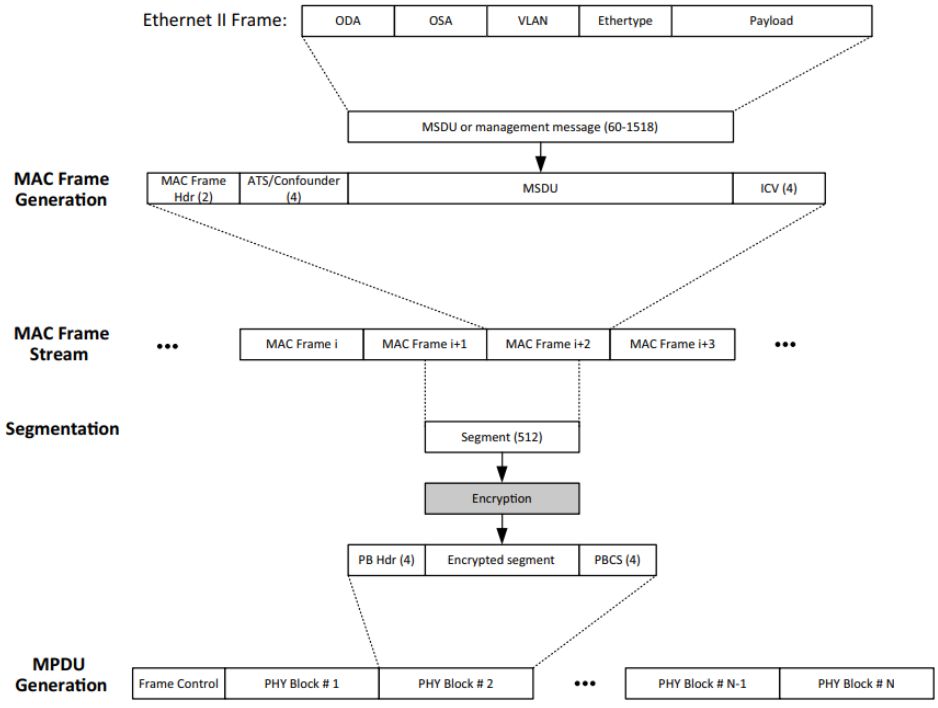

Figure 2.7: MAC Data Plane Overview (taken from [4])

This could be the case of the HomePlug Green PHY protocol applied to the EV

PLC modem project, since an eventual **Host MCU** located in the EVSE could exchange data in that format. Then it is up to the architecture to extract information in Ethernet format and make them compatible to HPGP in order to let the information be exchanged through the power line to reach the EV.

As it can be stated from figure 2.7, once the data in Ethernet format join the HPGP architecture through the Convergence Layer they become **MAC Service Data Units (MSDUs)**. Of course the **MSDU frame** is strongly related to the Ethernet one as it can be seen in figure 2.8.

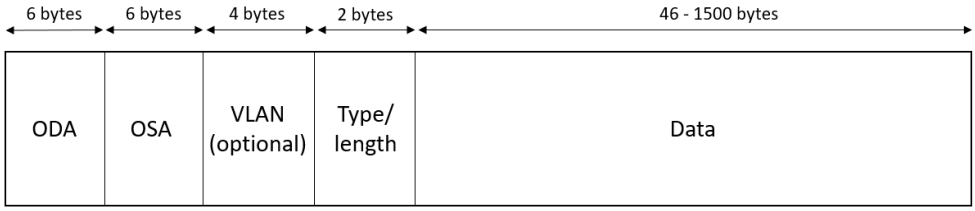

Figure 2.8: MSDU Format

where:

- **Original Destination Address (ODA)** is the 6-Octet MAC Address of the destination station(s) of the MSDU.
- **Original Source Address (OSA)** is the 6-Octet MAC Address of the station that is the source of the MSDU.
- **Optional VLAN Tag** is the 4-Octet VLAN Tag field that is formatted as described in IEEE 802.1Q.
- **Ethertype/ Length** is the 2-Octet Ethernet II Type/Length field.
- **Data** is the 46–1500 Octets of transported data.

So in transmission, the MAC layer has to convert **MSDUs** and management messages into **MAC Protocol Data Units (MPDUs)**. After that the MPDU will pass through the other main component of the baseband that is the PHY layer, which will convert it in a **PHY Protocol Data Unit (PPDU)**. The passage from MSDU to PPDU will be detailed in the following lines of this section.

Coming back to the formation of the MPDU, as depicted in figure 2.7, it is achieved through the application of the 4 steps in the following paragraphs.

2.6.1 MAC Frame Generation

In this first step, the aim is to generate a **MAC frame** starting from the MSDU frame or management message that has been derived from the Ethernet format, then it is encapsulated with further MAC layer information that are going to be described in the following lines. The structure of the MAC Frame will be the one in the following figure 2.9:

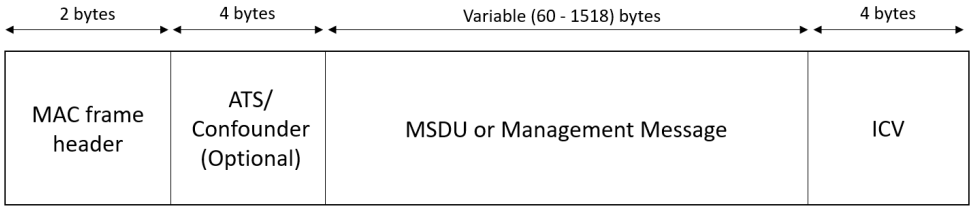

Figure 2.9: MAC Frame format

The fields in the MAC Frame are as follows:

- **MAC Frame Header** is a 2-octet field that consists of a 2-bit MAC Frame Type and 14-bit MAC Frame Length. The last one specifies the MAC Frame length in octets (bytes), without considering the 2-octet MAC Frame Header and the 4-octet **Integrity Check Value (ICV)** but including the **Arrival Time Stamp (ATS)** or the **Confounder** if one of them is present. The 2-bit field of the MAC Frame Type (MFT) indicates the kind of information that is contained in the MAC Frame. About that, there can be four possibilities:
  - **0b00**: MFT indicates the presence of a bit pad in the MAC Frame Stream. This is used when MAC Frame Stream has to be padded to generate a segment.
  - **0b01**: it indicates that the specified MSDU has not a correspondent ATS.
  - **0b10**: the MFT highlights the presence of the MSDU along with an associated ATS.
  - **0b11**: indicates the presence of a MAC Management Entry with associated confounder.
- **Arrival Time Stamp (ATS)** is a 4-octet field that contains the **Network Time Base (NTB)** at the time when the MSDU arrived from the Higher Layer Entity. This value is used by the receiver in order to provide a fixed latency within the power line network.
- **Confounder** it is a 4 bytes pseudorandom value. It is present whenever a MAC frame contains a management message and is used to render identical

messages as different cipher texts. This enhances the security by defeating recognized cipher text attacks.

- **MSDU or management message** is the payload of the MAC Frame. The contents of the MSDU or management message are not modified by the MAC Data Plane. In this context the word MSDU means the data content of the information, while a management message indicates a bunch of control information that are useful for a good communication.
- **Integrity Check Value (ICV)**: it is a CRC-32 computed over a MAC Frame. A **Cyclic Redundancy Check (CRC)** is a common method that is used in communication networks in order to discover if the transmitted data have been corrupted during the communication. This topic plays an important role in the description of the signals that have been chosen as input for the **Monitor IP** so the complete explanation of CRC can be found in chapter 4. The ICV does not cover the MAC Frame Header, ATS (if present), or confounder (if present). CRC-32 is computed using the following standard generator polynomial of degree 32:

  $$G(x) = x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1$$

2.6.2 MAC Frame Streams

This step involves **grouping of MAC Frames** into queues based on destination, so that MAC frames which are belonging to the same stream of information for the same receiving STA are put together in the same stream. Each stream associated with a MAC Frame depends on the following fields:

- **Destination Terminal Equipment Identifier (DTEI)** that is determined based on the ODA of the MAC Frame. Within an AVLN, each power line station is provided with an unique **Terminal Equipment Identifier (TEI)** by the CCo. The mapping from each station's MAC address with their corresponding TEIs is provided by the CCo to all stations in the AVLN. This information is used to determine the DTEI associated with the MAC Frame from the ODA.
- **Link Identifier (LID)** associated with the MAC Frame. In general are used to identify various Links within an AVLN.
- Whether the MAC Frame contains **MSDU** (i.e., data) or **management messages**.

So the **MAC layer** is responsible to verify the values of DTEI and LID for each MAC frame, in this way it can be able to recognize the "segments" of information

dedicated to a specific receiver and group them together. Then if the content of the MAC frames are MSDUs, that stream is defined **data stream** while if it is composed mainly by management messages it is a **management stream**

2.6.3 Segmentation and formation of the MPDU

The **Segmentation process** is highlighted in the following figure 2.10.

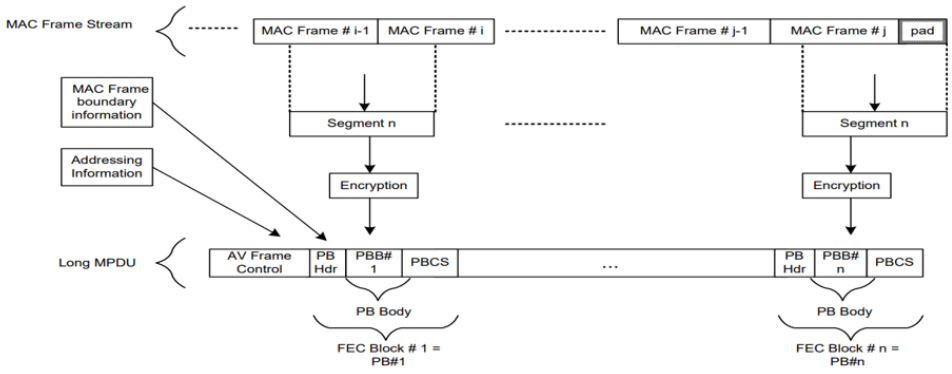

Figure 2.10: MAC segmentation and MPDU generation (taken from [4])

In practice, each MAC frame stream is broken down into segments made up of 512 bytes that are created every time that there is enough data (bytes) to form a complete segment. Since the stream by definition is a continuous flow of bytes, the segments can contain parts of different MAC frames which are variable in size because an MSDU or management message can have variable size at its turn.
Then for everyone of these segments, the MAC highlights the point in which a MAC frame begins in the segment and this information is then transmitted to the receiver in the **PHY Block Header (PB Hdr)** so that the receiver will be able to extract correctly the individual MAC frames from the segments that have been received. The PB Hdr also transmits the **Segment Sequence Number (SSN)** which is a value that starts from zero and increases for each generated segment. It is helpful for the receiver since it allows a better detection and handling of eventual out of order segments or duplicates.
At the end of the process, each segment is encrypted and then placed in a so called **PHY Block (PB)** that includes the data bits of the **payload** at the PHY layer. Each PB is composed by:

- a **PB Header** which indicates the SSN, the offset of the first MAC frame that is present in the segment and if the content is data or management stream.
- a **PHY Block Body (PBB)** which is the informative content.

- a **PHY Block Check Sequence (PBCS)** which is used to verify the integrity of the information at the receiver by mean of a CRC code.

At this point an MPDU is formed and it will have the bits composition presented in figure 2.11.

Figure 2.11: General MPDU format

However there are several different types of MPDU that are mainly distinguished by their functionality and carried information, above there are the most important typologies among them:

- **Beacon MPDU** contains the information that are necessary for the CCo to control all of the stations in the network making them to be synchronized. In particular this information consists mainly of the **Central Beacon** which represents a bunch of information on the **CSMA allocations**, explained in chapter 3 when describing the MAC layer architecture.
- **Start Of Frame (SOF) MPDU** has the role to transmit **data** and **management** information to one station of the network or more.
- **Selective Acknowledgment (SACK) MPDU** carries information that are received by the station that previously has transmitted an SOF or RSOF MPDU to another one. These information provide a **reception status** of the transmitted data packet.
- **Sound MPDU** is a kind of frame that is used by the CCo to **estimate** and **find** stations in the network.

Depending on the type of MPDU that is analyzed, there can be distinguished slightly different formats for the MPDU. As an example, in figure 2.12 can be denoted the specific format for a **Beacon MPDU** while in figure 2.13 is depicted the **SOF MPDU** one.

Regarding the Beacon MPDU, in its Frame Control field there can be found two important information that are the **Beacon Time Stamp (BTS)** which contains

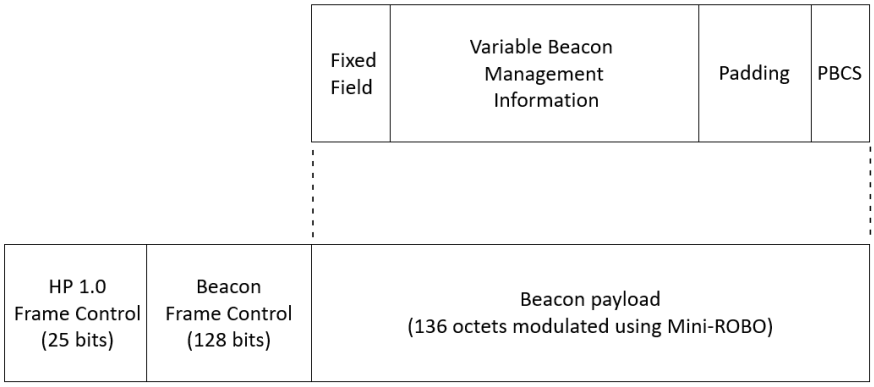

Figure 2.12: Beacon MPDU format

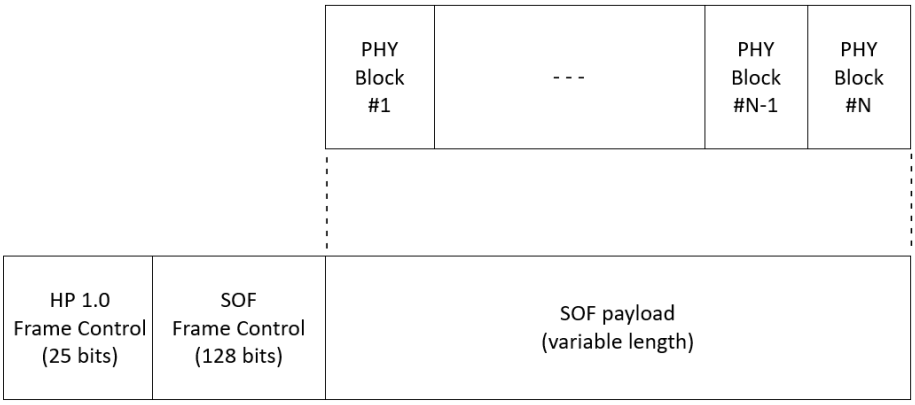

Figure 2.13: SOF MPDU format

the 32-bit time value that indicates the instant at which the transmission of the Beacon started and the **Beacon Transmission Offset (BTO)** which tracks the **AC-line cycle** time variations that so are translated into a variable offset to be applied at the BTS value. These two parameters are useful for the **Network Time Base Synchronization (NTB)** which consists in the correction of the clock signal that has been transmitted by the CCo and received by the other stations. The NTB will be explained in more depth in chapter 3 when describing the architecture of the MAC layer.
The payload portion of the Beacon MPDU frame is formed by 136 bytes of data where there is a **Fixed Field** composed by 92 bits that carries information such as the **Network ID (NID)** and the **Source Terminal Equipment Identifier**

(STEI) which is a numerical identifier of the station that has transmitted the Beacon. Then there is a variable number of **Management information** bits that is subdivided in units called **Beacon Entries (BENTRIES)**. In particular a BENTRY is a piece of information which has a length that is specified by a field of 1 byte named **Beacon Length (BELEN)** and the number of BENTRIES for every Management Information field is specified by a further 8-bit field named **Number of Becon Entries (NBE)**. In addition there is another field named **Beacon Header (BEHDR)** which is either composed by 8 bits and specifies the kind of information carried by a specific BENTRY.
The last field of the payload is the **Beacon Payload Check Sequence (BPCS)** that is a 32-bit field that is used to check the correctness of the payload through a **Cyclic Redundancy Check (CRC)** computed on all the payload bits with exception of those of the BPCS itself. The remaining bits among the 136 bytes which are variable because their number depends from the length and the number of BENTRIES, are padded with zeros.

As it can be seen from figure 2.13, the SOF MPDU format is slightly different with respect to the one of the Beacon that has been just analyzed and it is in practice the standard MPDU formation that has been described previously in figure 2.7. As a matter of facts in this case the payload is of variable length and it is composed by a finite number of PHY blocks, where each PHY Block is composed by 136 or 520 bytes. As it can be noticed, here the difference between the formation of the Beacon MPDU and the SOF one is that the first is just composed always by a single 136 bytes PHY Block, while the SOF MPDU is composed by more PHY Blocks with variable length.
The two PHY Block formats are depicted in the following figure 2.14.
Where the fields composing the formats have been explained previously.

2.6.4 From MPDU to PPDU

In transmission, the MPDU represents the format of the frame at the output of the MAC layer that is in charge of providing to the PHY the information to encode and transform in **PHY Protocol Data Unit (PPDU)**, so the frame format that effectively is transmitted on the **power line**. Then the PPDU in the analog domain reaches the receiver AFE which at its turn, making use of its **ADC**, get samples of the PPDU that are decoded at PHY layer level and the result of this operation will represent the MPDU at the receiver side that will be delivered to the receiving MAC layer. As a reference of this passage consider the following schematic reported in figure 2.15.
The single entity of the analog PPDU is named **OFDM symbol**. An OFDM symbol is the result of the modulation that is performed by the blocks that compose

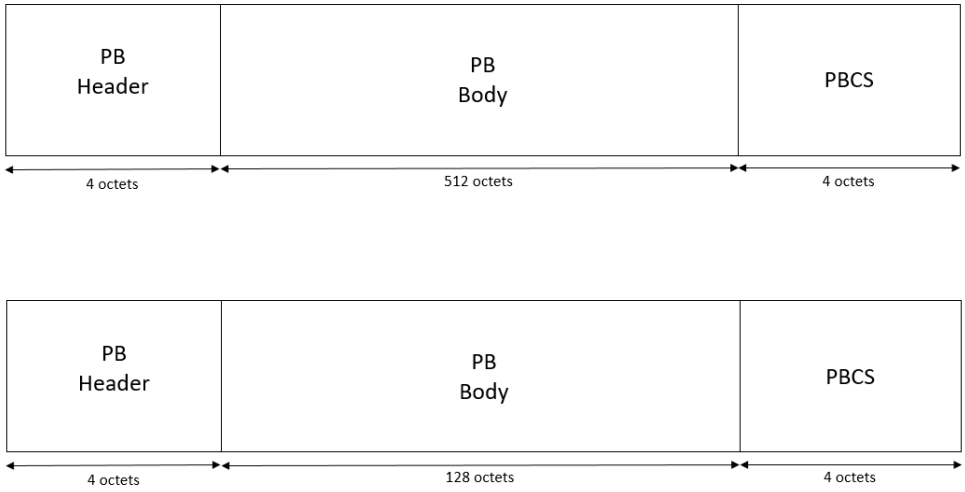

Figure 2.14: PHY Block formats

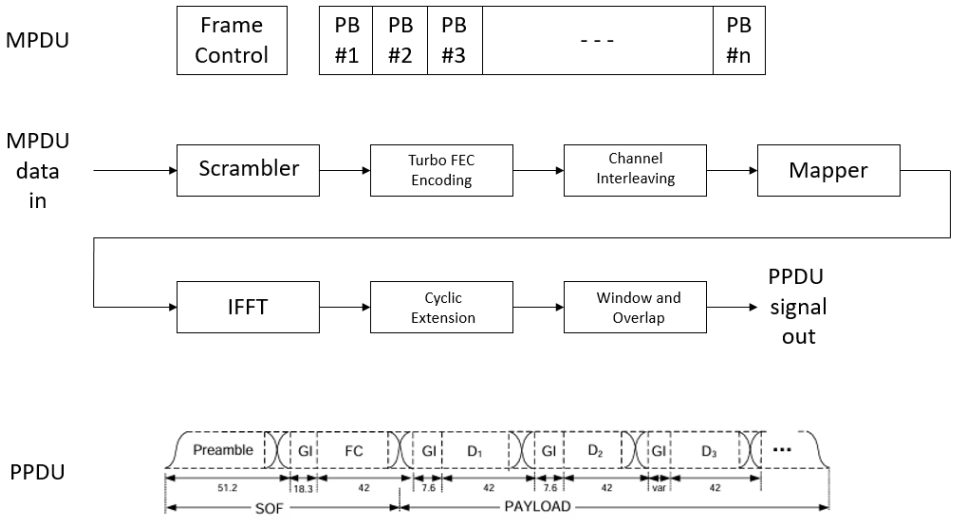

Figure 2.15: MPDU to PPDU encoding

the architecture of the PHY layer which will be deeply explained in the dedicated chapter 3. The different portions of the PPDU frame (Preamble, Frame Control and Payload) are made up of one or more of these symbols and every symbol is based on a sampling clock of 75 MHz introduced by the DAC of the AFE. In figure 2.16 it is depicted the timing of an OFDM symbol where the correspondent timing parameters are reported in the table 2.3

As it can be seen by looking both at the table 2.3 and the figure 2.16 the period of

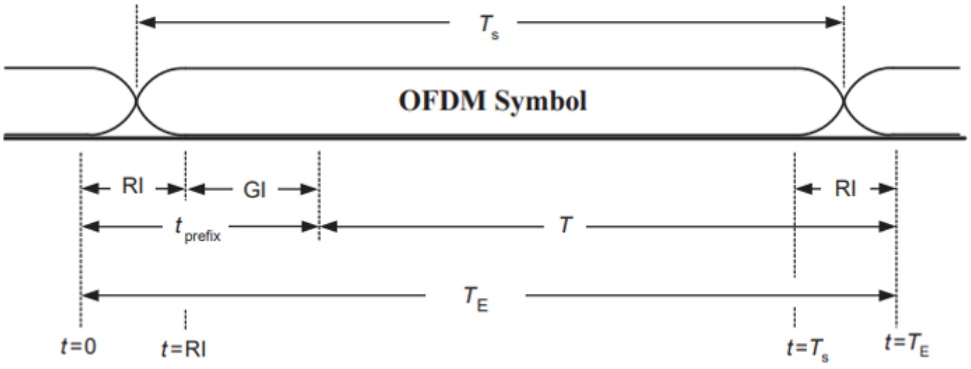

Figure 2.16: OFDM symbol timing (taken from [4])

Symbol	Description	Time Samples	Time (μs)
T	IFFT interval	3072	40.96
t_{prefix}	Cyclic prefix interval	RI + GI	4.96 + GI
T_E	Extended symbol interval (T + t_{prefix})	T + t_{prefix}	45.92 + GI
RI	Rolloff Interval	372	4.96
T_S	Symbol Period	3072 + GI	40.96 + GI
GI_{FC}	Frame Control Guard Interval	1374	18.32
GI	Payload symbol guard interval, generically	417, 567, 3534	5.56, 7.56, 47.12
GI_{SR}	STD-ROBO Payload symbol GI	417	5.56
GI_{HR}	HS-ROBO Payload symbol GI	417	5.56
GI_{MR}	MINI-ROBO Payload symbol GI	567	7.56

Table 2.3: OFDM symbol characteristics

a symbol is equal to the time interval of the **Inverse Fourier Fast Transform (IFFT)** which is a key passage in the encoding process summed with a **Guard Interval** which is needed by the receiver to distinguish correctly the symbols among them. The IFFT operation requires 3072 samples, so from the moment that the sampling clock is of 75 MHz, this means that:

$$T = 1/75e6 * 3072 = 40.96\mu s$$

In addition to that, as it has been said, a Guard Interval must be introduced which changes in number of time samples depending if it is specified on the Frame Control or the Payload, as well as if the used ROBO mode is standard, high speed or mini.

Chapter 3

Electric Vehicle Power Line Communication Modem

The **EV PLC modem** represents the context in which the Monitor IP treated in chapter 4 has been inserted. In this chapter its system architecture will be treated, starting from an application point of view for going through the singular digital and analog components of which it is composed, explaining their role in the entire final **System on Chip (SoC)**.

As it has been briefly introduced in chapter 1, the EV PLC modem is an IP that has been developed by ST Microelectronics for the exchange of data between an EV and an EVSE using a power line communication. There are several standards that can be adopted for managing a PLC, in this specific case for reasons that have been explained in chapter 2, the modem is compliant with the **HomePlug Green PHY (HPGP)** standard of power line communication. This means that the information will be packed in **HPGP frames** by the monitor, exchanged between the EV and the EVSE though the power cable, unpacked again by the receiving monitor and then processed by the host processor.

Of course since the exchange of information is bidirectional, a modem must be present in both the EV and the EVSE. However, it is not mandatory that the **EV PLC modem** developed by ST must be present in both, the fundamental aspect for having a proper exchange of information is the **communication standard** that has to be supported by all of the involved devices. As an example, if the EV uses the ST modem and the EVSE uses a Qualcomm one, if also the latter supports HPGP protocol, information can be exchanged without problems. As a matter of facts, the EV PLC modem has been thought primarily for being installed inside the EVs because an initial ST Microelectronics' s market survey revealed that the

most of the possible customers interested in the product were **car manufacturers**. This was sufficient to make the project starting, because as it has been just said, it is not necessary to have that IP on both the sides of communication.

Another preliminary concept that must be explained, is that this device can be used both in an **AC or DC charging system**. In an AC Charging system the AC power coming from the grid is converted in DC power for the battery directly by the **AC-DC converter** inside the EV while in the case of a DC Charging system, the conversion is made at the level of the EVSE that than delivers DC power to the battery of the EV. A DC Charging system allows to charge the battery inside the EV more rapidly with respect to the AC counterpart. This because the more is the power that is provided, the faster is the charging and to do that there is the need of a large AC-DC converter that for space reasons is possible to adopt only in the EVSE. Actually, both the kinds of charging are adopted, this because despite the AC Charging is slower that the DC one, it is cheaper to realize for a company because it does not need a huge and costly AC-DC converter to be integrated in an EVSE and can also be performed at home, by plugging the EV in the home grid.

As introduced before, an EV PLC modem is compatible with both the way of charging but since its duty is to modulate and demodulate packets of HPGP information, it is more "meaningful" if it is present in a DC Charging system. This because the **charging curve** for the DC case is initially high for allowing an initial fast charge, than it decreases more or less slowly dependently from the conditions and the kind of battery in order to avoid overheating. This variation in the charging rate during the period of charging requires a constant exchange of information between the **Battery Management System (BMS)** of the EV and the **host processor** of the EVSE, this is why managing a PLC is strongly needed in this case. For the AC Charging, from the moment that the charge relies on the EV converter, this usually handles a power level that does not vary but remains constant with time, regardless the conditions of the battery. In this case it can be referred as a flat charging rate for the EV battery, and there is no need of an intense information exchanging between the two actors. Anyway a PLC can still be useful also in that case because a user may want or have the necessity to charge its EV in a public smart grid that makes use of AC EVSEs that as their DC counterpart have to deal with the billing information of the user that are related to the actual state of his EV battery.

3.1 An high level overview on the project architecture

This section represents an overview on the **Electric Vehicle Power Line Communication modem (EV PLC modem)** architecture realized by **ST Microelectronics**. The device can be introduced with the following figure 3.1 taken from the **EV_PLC_ADD94** [9] team document.

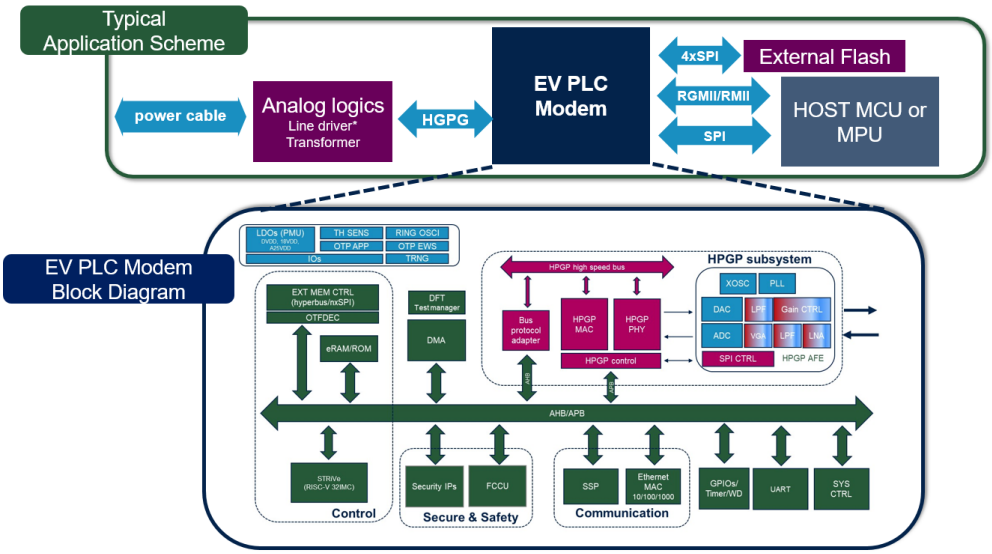

Figure 3.1: EV PLC system concept (taken from [9])

As it can be seen, there can be distinguished two diagrams in the picture. The upper one, as depicted, represents the **Typical Application Scheme** of the project, as a matter of facts it figures the high level interaction of the projected device with the other components of the communication system while the **EV PLC Modem Block Diagram** is an in deep representation of the components of the developed modem.

For a better understanding of the application scheme, as an example the reader may imagine that the depicted EV PLC modem in the previous figure is located inside an EVSE and an hypothetical EV can be located on the left of the figure and connected through a power cable to the developed **EV PLC modem**. Then the system exchange information between EV and EVSE by mean of the same cable that carries the power for charging the EV battery with the use of a proper standard of communication represented by the **HPGP** that guarantees signal integrity and encoding for a secure exchanging of data. In addition to that some analog components such as a **transformer** inserted in an **Analog logics** set, are

necessary for adapting voltage, current ad load impedance to values supported by the standard.
In conclusion, as it has been said, the role of the modem is to exchange **HPGP data frames** with the EV by mean of its internal architecture that will be explained in the next sections, furthermore it will exchange data with an **Host MCU** that thanks to the demodulated data obtained from the modem it will get important information for the charging of the EV such as the **battery status**. On the other hand the MCU can manage the charge process programming the modem by sending to it data through an **SPI** interface and a **RGMII/RMII (ethernet)** interface. The modem can also rely on an **External Flash** for exchanging data making use of other 4 **SPI** interfaces.

The second scheme of figure 3.1 represents an in depth view of the **EV PLC modem**. On the upper right corner of the picture the **HPGP subsystem** can be noticed. It is one of the most important parts of the entire system because it allows to manage the **HPGP data frames** letting the information to be exchanged between the power line and the **digital core** that is represented by the blocks highlighted in green in figure 3.1.
Regarding the **HPGP subsystem**, it has been developed by an american company which helped ST Microelectronics providing one of their dedicated IP for handling the HPGP format that has been simplified in order to support only PLC communication through HPGP protocol (The **Zigbee interface** has been removed). Then Greenvity and ST MIcroelectronics cooperated for making the HPGP subsystem compatible with the rest of the ST **digital core**.
The HPGP subsystem is composed by an analog front-end called **HPGP AFE** that in transmission is responsible of converting the digital HPGP frame coming from the **digital baseband** in an analog signal and adapt it in order to make it compliant for the transmission on the power line. On the contrary when receiving, the AFE converts the analog signal coming from the power line to a digital signal that will be manipulated by the baseband.
The **digital baseband** is mainly composed by the **Physical layer (PHY)** and the **Media Access Layer (MAC)** components. The basic role of the **MAC** is to format the digital data coming from the **digital core** in fixed-length frames for transmission on the channel while the **PHY** must perform **Forward Error Correction (FEC)**, manipulate these MAC frames and map them in **OFDM symbols**, generating time domain waveforms. An adequate study of the baseband components will be carried on in section **??**.

The green blocks in figure 3.1 represent the **digital core** of the system and have been developed entirely by ST Microelectronics. At its turn, the digital core is composed by many IPs and subsystems. Since at this point the target is to give

just a generic but complete overview of the system, these IPs will be listed and briefly explained, taking into account that in the following sections they will be treated more in detail:

- **Control Unit (CU)**: core module based on a **RISC-V** microprocessor embedded in the EV PLC.
- **Safety IP**: it is a **Fault Collection Control Unit (FCCU)** that monitors the device functionality according to the safety requirement.
- **Security IPs**: they are required for supporting **encryption** and **decryption** functions as well as authorizing the executed code and implementing security on the data transfer by the communication IPs.
- **Communication IPs**: EV PLC supports **Ethernet** and **SPI** formats for data exchange with the host processor.
- **EV PLC memories**: several top level memories are required to implement an HPGP modem functionality.
- **Top level digital IPs**.
- **Top level analog IPs**.

3.2 HPGP subsystem

The HPGP subsystem represents one of the most important units of the entire architecture of the Modem IP because, as it has been said previously, it is in charge of **modulate** and **demodulate** the data to and from the HomePlug Green PHY format, acting as intermediary between the digital signals of the **digital core** and the information in **HPGP symbols** that is exchanged through the power line. To have a good understanding on the HPGP subsystem architecture is also important for the development of the **HPGP Monitor IP** that is treated in chapter 4 because some of its blocks contain signals that are exploited also in that new IP that is needed to make considerations on the correctness of the communication.

In the following sections will be analyzed the architectures of the main blocks that compose the entire HPGP subsystem that has been provided to the ST team by the already mentioned external company, taking into account that the schemes that will be proposed as well as the correspondent explanations will not go too much in details because of the **confidentiality** about the project that both ST and the external company asked.

3.2.1 HPGP Analog Front End (AFE)

The **AFE** is an analog IP combining all the Power Line Communication (PLC) functions to meet the requirements for interfacing HomePlug Green PHY to a suitable digital architecture. In the following figure 3.2 there can be denoted an high-level schematic of its architecture.

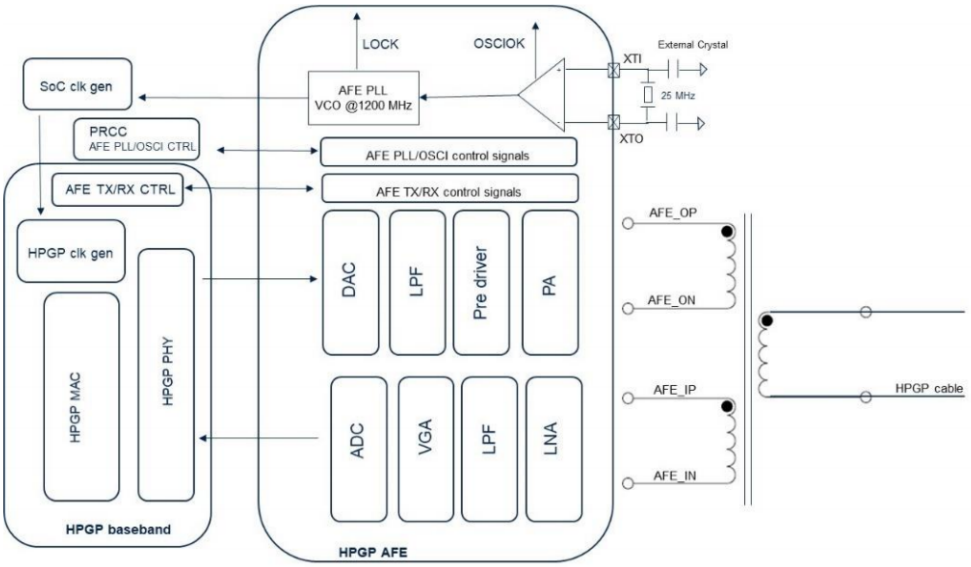

Figure 3.2: HPGP AFE high-level architecture (taken from [9])

The first thing that can be denoted by looking at figure 3.2 is the presence of two chains of units inside the AFE architecture. As a matter of fact there is a **TX chain** that is activated when the Modem in which the HPGP subsystem is located has to transmit some communication frames to a receiving one. That chain involves a **DAC** which converts 12-bits samples into an analog signal that then is filtered by mean of a 28 MHz fixed bandwidth **Low Pass Filter (LPF)** and amplified using a **Pre-Driver** and a class AB **Power Amplifier (PA)** which is designed to drive 100 Ω loads up to 4 V peak to peak amplitude. It has the role to increment not only the gain but also the power of the signal since it has to be transmitted on a power-line that, even if it is optimized, it will lead to an unavoidable **attenuation**. Both the Pre-Driver and the PA provide a gain control range of -9 dB to +9 dB and the DAC has a fine attenuation range of 0 dB to -12 dB in 0.5 dB steps to meet transmit mask requirements.
RX chain has been designed to operate with a minimum impedance of 100 Ω with modulation frequencies up to 28 MHz to meet HPGP specifications and making analogous considerations as before, it is activated when the correspondent Modem is receiving frames from another one, so this time when a frame is received there

will be the need of a **Low Noise Amplifier (LNA)** that provides a voltage gain from -12 dB to 24 dB because it is expected that the received frame has been attenuated by the power line and so the goal is to amplify the analog signal adding less noise as possible. Later on in the chain, the analog signal passes through a **Low Pass Filter (LPF)** for eliminating high frequency noise and then through a **Voltage Gain Amplifier (VGA)** that has the role to set the received analog signal to the correct dynamic range before being processed by a **12-bit ADC** of pipeline type as a final unit in the chain which operates at a sampling rate of 75 MHz. The total VGA gain range is -18 dB to +48 dB. That final operation is important because it allows to have an analog to digital conversion with a good resolution, without distortions and having a nice **Signal to Noise Ratio (SNR)**. It has to be denoted that in figure 3.2 there is a block in the AFE named **AFE TX/RX control signals** that exchanges signals with an analogous block in the Baseband which is called **AFE TX/RX CTRL**, this means that some signals of the AFE, as the gain of the VGA, are given by control signals coming from the Baseband. In the case of the VGA its gain is refined by the gain control signal at the output of the **AGC** unit in the PHY of the Baseband, other similar cases are represented also by the gains of the LNA, LPF and PA for the transmitting chain.

Other elements of the AFE that can be observed by looking at figure 3.2 are those regarding the **clock generation and management**. At the start-up of the system it is needed a 25 MHz clock, generated by an **external oscillator** that once it is stable it provides to the Modem an **OSCIOK** signal that is stored in a Modem configuration register. When this signal is read from the register, it means that the power up phase of the device is finished, so the clock source is switched from the ring oscillator to the output of a **Phase Locked Loop (PLL)** that is locked to the 25 MHz to generate on-chip sampling clocks. Moreover the clock generated as an output of the PLL of the AFE is sent to the **SoC clk gen** of the Modem that is a logic that generates all the needed clock domains as well as those that are needed also for HPGP Baseband.

Regarding the other features of the AFE, they have to be mentioned the **Test Pins** for evaluating the receive and transmit filters and the **external transformer** which is required for connecting RX and TX line to the cable. Furthermore it has to be said that the HPGP functionality shall be **half duplex** so that TX and RX will never be working in parallel in functional conditions. However, the **full duplex** mode shall be supported only for functional debugging scope.

3.2.2 HPGP Baseband: PHY layer

At this point the PHY layer which is one of the three main modules that compose the architecture of the Baseband along with the MAC layer and the HPGP clock generation unit has to be analysed. This macro-component of the HPGP subsystem Baseband is in charge of **modulating** the digital data into the HPGP format in transmission and **demodulating** it again to digital format when receiving. Its high level architecture block diagram is showed in the following figure 3.3.

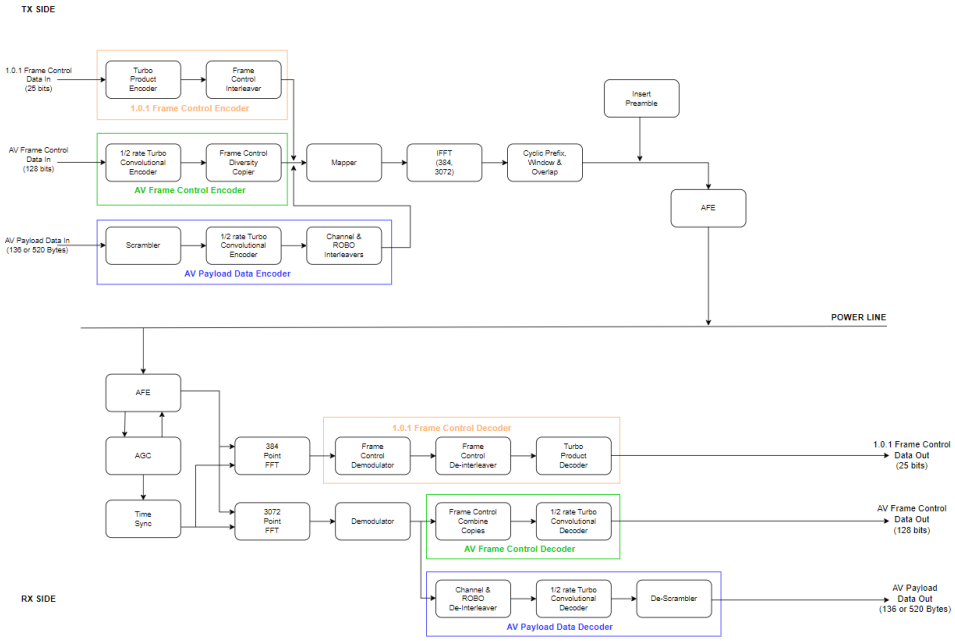

Figure 3.3: HPGP PHY high-level architecture

Since the description of the frame and so on what an **OFDM symbol** is as well as a **PPDU** has been just explained in chapter 2, now the focus is on the blocks that compose that architecture and their function on the modulation/demodulation chain of the PHY layer. One of the first things that can be noticed by looking at the figure 3.3, is that the PHY architecture is divided in two complementary groups of modules, one that regards the modulation of the digital data and their transmission on the power line denominated **TX side** and the other, the **RX side**, that demodulates the received symbols on the power line to obtain digital data to deliver to the MAC layer. As a matter of fact, on the transmitter side, the PHY receives the digital data from the MAC and then, depending on the kind of information, the bits can enter three different **encoding chains** so that they can be in contact with different types of modules.

Frame Control Encoding

The encoding process for the 128 Frame Control bits involves the modules depicted in the following figure 3.4.

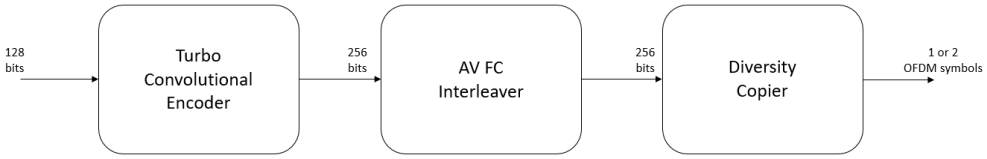

Figure 3.4: Frame Control Encoding chain

As a first step, the 128 bits of the 128 Frame Control bits enter the **Turbo Convolutional Encoder** reported in figure 3.5

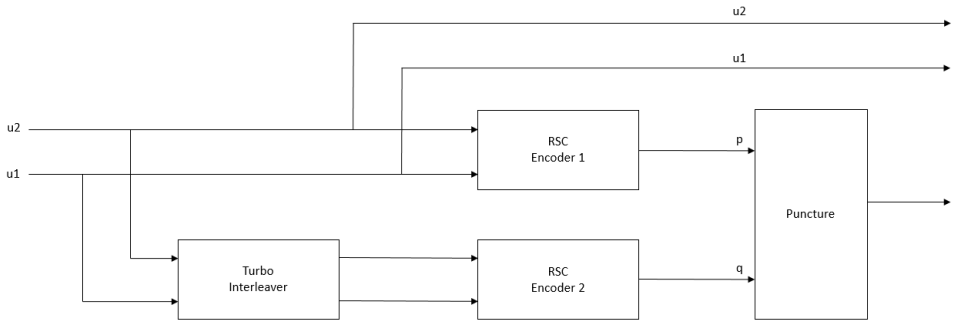

Figure 3.5: Turbo Convolutional Encoder Block Diagram

which encodes those bits into 256 coded bits. As described in the HPGP standard [4] that the external company used as a core reference for the design of the entire HPGP subsystem architecture, this module takes as inputs the streams of bits **u1** and **u2** which represent the sequence that must be codified, indeed the first bit of the Frame Control is mapped to u1, the second to u2 and so on. Then the two signals enter the **Turbo Interleaver** that has the role of shuffling the data in input, changing the order in which they are presented to the next unit. Then there are two **Recursive Systematic Convolutional (RSC) Encoders** where the first one takes in input the original streams of bits and the other the stream that has been shuffled by the Interleaver. The two encoders generate two sequences **p** and **q** of parity bits by mean of a **Recursive Encoding Process** that is reported in the scheme of figure 3.6.
In practice there are **three states** represented by s1, s2 and s3 which are implemented making use of three registers that store the results of the sum between the

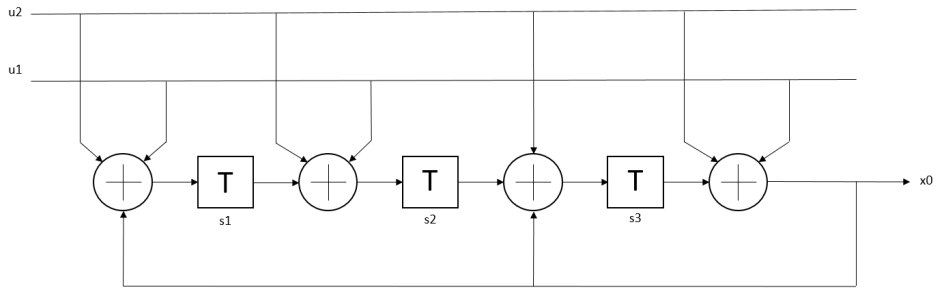

Figure 3.6: Scheme of the RSC encoder

bits in u1 and u2 along with the output x0 and the value stored in the previous register (state). So for every bit in input, a new state (so a new register value) is computed and every new bit is added to the previous values that are kept in the registers and that contributes to compute the output x0 which at its turn is sent as a feedback to some of the four adders. It is called a "recursive" operation since the output is based not only on the actual input values but also on the previous ones. Then the output of each encoder is sent to a so called **Puncturing Pattern** which is in charge of **eliminating some parity bits** in order to decrease the overall quantity of transmitted data, making a sort of compromise between the **code robustness** and **band efficiency**.
At the end of the process, the output of the Turbo Convolutional Encoder will be characterized by the u1 and u2 streams named **systematic sequences** (so they are the information bits) and by the single stream in output from the Puncturing Pattern unit which characterizes the **parity bits** of the transmission.

Coming back to the process of Frame Control Encoding, the output of the Turbo Convolutional Encoder will enter a **Channel Interleaver** unit that plays a very important role in the transmission because it **interlaces the bits** after they have been encoded by the TCC, so that if they are corrupted or lost during the transmission process, the **error is distributed** so that the decoder can correct them in an easier way. In practice, the information bits are divided into **four equal sub-blocks** where every block has **k/4** bits with k total number of information bits and the same for the parity bits that are divided in four sub-blocks where each of them has **(n-k)/4** bits with n total number of encoded bits.
All of these sub-blocks have a **matrix organization** so that for instance the information bits can be considered to be organised in k/4 rows and 4 columns, so every column represents a sub-block of bits. The bits are not read from the matrix row by row but the Interleaver "jumps" a certain number of rows determined by a

valued named **Step Size**, that is the same both for the information and the parity bits although for the latter it is specified a certain **offset** which makes the reading of the matrix to start from a different row for the parity bits with respect to the information ones. When the reading of the matrix is finished, the process **wraps around** and starts again from the beginning, then it will not stop until all the coded bits will be read from the matrix. In addition to that, the bits are again shuffled and organised in new sub-blocks every time that 4-bits are read in order to give a further degree of bit shuffling.
This process is useful not only because the errors are spread so that they can be corrected easily, but also because in this way **parallel decoders** in the RX side can be used and this thing speeds up the decoding process, increasing the performances.

So after that the 256 encoded bits have been interleaved, they become the input of the **Diversity Copier**, which has the only role of **QPSK mapping** these bits onto the used carriers. That means that the interleaved bits are transmitted with an address offset of 128 between the **in-phase (I)** and the **in-quadrature (Q)** channels. Since the carriers in the frequency range supported by the HPGP are 1155 and in a transmitted frame there is just one **FC OFDM symbol**, each bit is copied at least 7 times and some bits even 8 times for each carrier. This last feature has to give the idea of the robustness of such kind of communication protocol.

A last consideration on the Frame Control Encoding must be done for the **1.0.1 Frame Control** that is specific in the case of the communication between an HPGP device and a HomePlug 1.0.1 one. Since it is a particular case and for sake of brevity, the correspondent encoder will not been explained in details. However it has to be considered that the most relevant changes are relative to the number of FC bits that are 25 in the 1.0.1 case instead of the 128 that has been treated, moreover a **Turbo Product Encoder** which implements a different logic for encoding the bits has been used but its accurate description goes beyond the goals of this thesis.

Payload Encoding

For the **Payload Encoding process** the process is very similar to the one described for the Frame Control, as it can be appreciated by looking at figure 3.7.

One of the differences with respect to the previous case is represented by the presence of the **Scrambler** unit which plays a similar role to the one played by the Interleaver but in practice it gives **random distribution** to its input bits while the Interleaver has a more deterministic approach. The Scrambler make the data stream to be **XOR-ed** with a repeating **Pseudo Noise (PN) sequence** that is

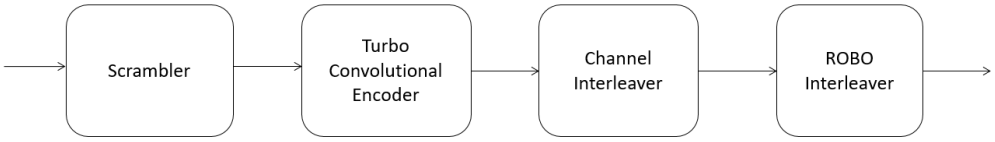

Figure 3.7: Payload Encoding block diagram

produced by mean of the following generator polynomial:

$$S(x) = x^{10} + x^3 + 1$$

In addition, all the bits in the scrambler shall be initialized to all ones when processing a new MPDU from the MAC layer. In the following figure 3.8, it can be found a representation of the scrambling process.

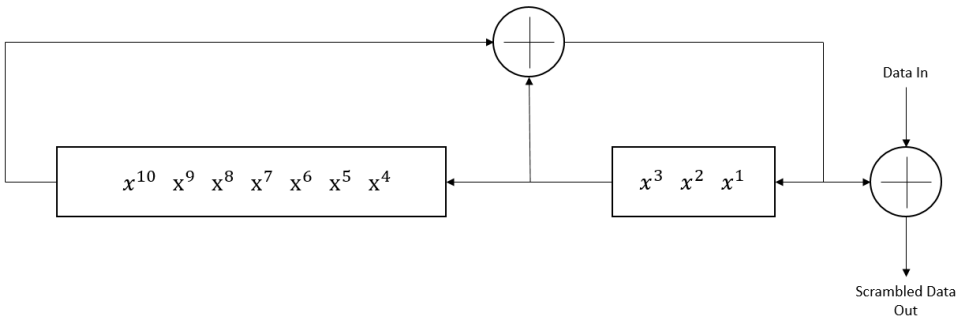

Figure 3.8: Scrambler block diagram

In particular the polynomials in the figure are implemented as **shift registers** so that the data which passes though them are delayed and then recombined together by XOR operation to get a random **Scrambled Data Out** value as output of the unit.

After that the payload data are processed by the Scrambler, they go in input to a further Turbo Convolutional Encoder followed again by a Channel Interleaver as in the case of the Frame Control bits encoding, with the only difference given by the number of the bits that is no more 128 but it is variable.

However this time at the end of the encoding chain there is a new block named **ROBO Interleaver**. Recalling that the definition of **ROBO modes** has been given in chapter 2, the role of this unit is to **manage the redundancy** and **data distribution on many carriers** in order to further increase the robustness of the

transmitted information and so to guarantee that the number of used carriers is a multiple of the number of **ROBO copies**.

Now that the three different encoding chains for the two versions of Frame Control and Payload have been explained, the attention can be shifted towards the modules that follows, taking as a reference figure 3.3.
As it can be seen, the three streams of data coming from the three encoding chains reunite as an input of a unit named **Mapper**. Without going too much in details so as not to make the discussion too verbose, the Mapper **sorts** the 3 typologies of frame bits, in such a way to give information to next modules in how to manage the incoming bit sequence.

As a matter of facts the next modules are the "core" ones since are those that are responsible of the **symbol generation**, which is the sub-unit that together with other OFDM symbols generates the **PPDU** which is the physical entity transmitted over the power line, as explained in chapter 2. So this process begins when the data are processed through an **Inverse Fast Fourier Transform (IFFT)** unit that in the case of **HomePlug 1.0.1** frame control data is based on 384 samples while in all the other cases, so both AV frame control and payload is on 3072 samples. Then a fixed number of samples is taken from the end of the IFFT and inserted at the front of the IFFT interval as a **Cyclic Prefix** to create an extended OFDM symbol that so is more robust.
After this operation the **Preamble** is inserted, which is a fixed sequence that is used for **frame synchronization** when the frame has to be received. In particular, the Preamble sequence depends just on the **station type** that can operate in **AV Only Mode** or in **Hybrid Mode** where in this last case there is also the presence of the HomePlug 1.0.1 frame control as it has been said when describing the HPGP standard in chapter 2.

After the described procedure, the symbols are converted from digital to analog by the AFE and then are transmitted on the power line. This thesis will not go further in the description of the blocks in the RX side, since the procedure is analogous to the one already provided for the transmitter case. The only two blocks that worth to be mentioned are the already cited **Automatic Gain Control (AGC)** module and the **Time Sync** one, where the first one is in charge of analyzing the received frame computing its **RSSI** value to have an estimation of its power so that, if the power is sufficient, the frame can be amplified (or even attenuated) in order to be set at the correct dynamic range for being properly demodulated by the PHY and this is done as a sort of feedback since the AGC communicates the **optimal gain** to the AFE that so modifies the dynamic of the received analog frame. Regarding the Time Sync module, it has the role of checking the correctness

of the preamble so that the demodulation procedure held by the PHY in receiving mode can be correctly synchronized.

3.2.3 HPGP Baseband: MAC layer

The **Media Access Control (MAC)** layer represents the "brain" of the entire HPGP subsystem. The MAC layer is maybe the most complex component of the HPGP subsystem and of all the Modem IP project because it has a lot of features and functionalities, most of them are beyond the main objective of the chapters 2 and 3 that is to explain the basic information needed to understand the project and the HPGP communication protocol in order to make the reader aware about what is behind the implementation of the **HPGP Monitor** introduced in chapter 4.

Taking as a reference the theory on the HomePlug Green PHY standard of communication explained in chapter 2, the MAC is in charge of performing many important actions that can be summarized in:

- **Manage the AV Logical Network** by selecting the **CCo** of the network through an algorithm.
- **Synchronize the communication** making use of the **Beacon MPDU**.
- **Format Data frames** into fixed-length entities.
- **Guarantee error free and in time** delivery of the data.

In figure 3.9 there is an high-level block diagram of the implemented MAC architecture.
It has to be recalled that the main objective of the **EV PLC Modem** is to receive and transmit HPGP data and exchange them with an **host processor** that will be in charge of exploiting these data for managing the smart charge of the vehicle. For this purpose are used the **Ethernet** and **SPI** protocols through the implementation of the **ETH GMAC** and **SSP** communication IPs of the Modem's digital kernel. Given that, looking at figure 3.9 can be noticed the presence of an **AHB port** which is used mainly by the MAC to exchange HPGP data with the above communication IPs. The **APB port** is available mainly for programming the HPGP registers.

The main unit of the architecture is the **Q controller**, a control unit that can be seen as the core of the MAC since it has to manage the **MAC Data Plane** so the conversion from an Ethernet Frame to an MPDU frame to be sent to the PHY in transmission and viceversa, as well as all the control signals that are necessary for the other modules in the MAC layer.

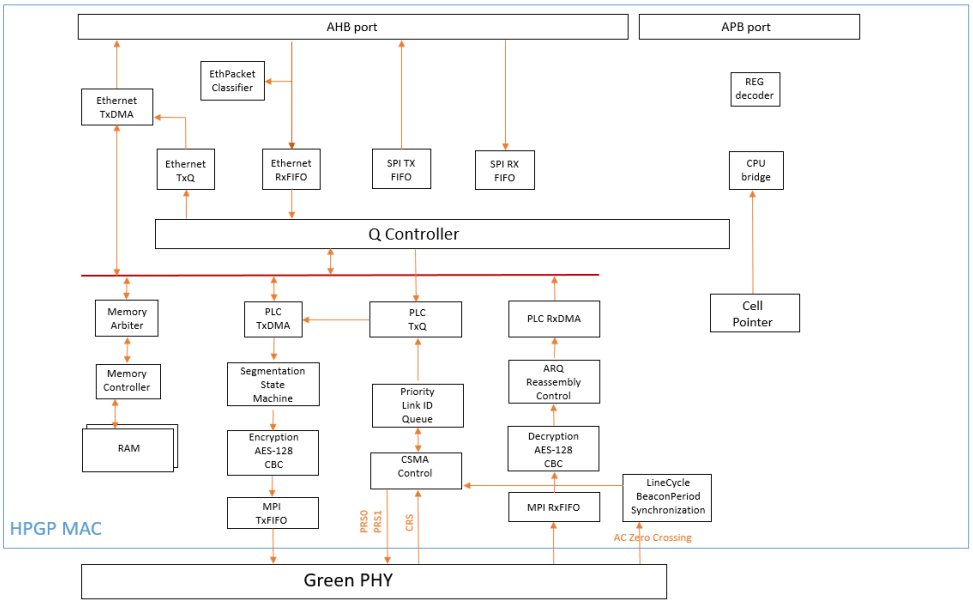

Figure 3.9: MAC layer block diagram

The Q controller has also another important function that is the one of **selecting the CCo of the network**. In an AVLN the CCo is chosen by an **auto-selection algorithm** so that every station in the network can decide to be the CCo basing its choice on the information that can get from the other STAs. This algorithm works so that if a station is the first one in a network, it automatically becomes the CCo, then if other STAs join the AVLN the decision is held on the basis on how many stations each STA can discover by mean of the **Sound MPDUs**. The number of discovered stations indicates the number of stations in the AVLN with which the station can directly communicate, so this criterion makes the STA with the best visibility in the entire network to be selected as CCo. At a parity of this property, making an analogous consideration, the STA that has the best visibility in terms of **discovered networks** has the highest potential for being the coordinator of these neighbor networks, so it is chosen to be the **CCo**.
In addition, The Q controller unit is responsible of the **Network Time Base Synchronization (NTB)** that is presented in the following figure 3.10.
As it can be denoted by looking at figure 3.10, the STA_clk is derived from the CCo_clk but the process leads to an offset that must be compensated, so by mean of the **Central Beacon** the CCo provides to all the STA the time stamp at which it has sent the clock signal. The difference between the delivered time stamp and the one of each STA that indicates the reception time of the clock allows to calculate this offset that so can be compensated.

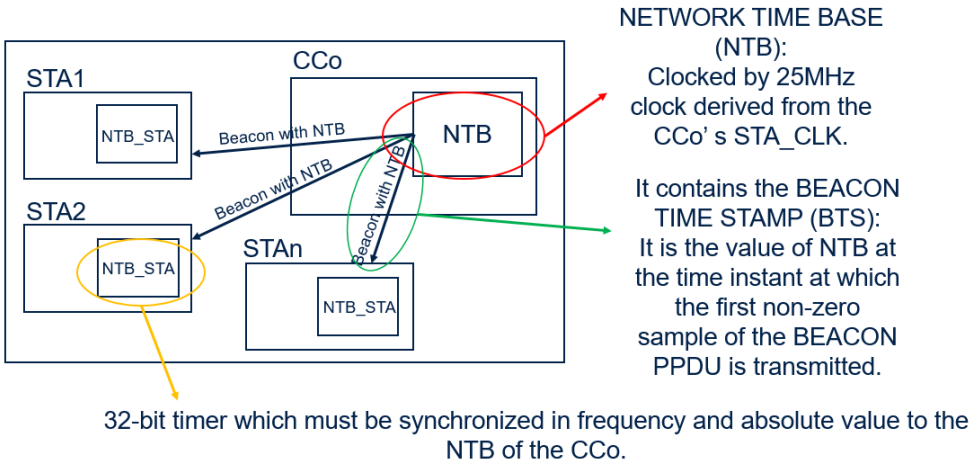

Figure 3.10: Network Time Base Synchronization

Always on the MAC Data Plane that has been deeply analyzed in chapter 2, the **PLC TxDMA** unit organizes the **MAC frame stream** while the **Segmentation State Machine** and the **Encryption AES-128 CBC** are responsible of the formation of the PHY block that will compose the payload of the MPDU fram sent to the PHY by mean of the **MPI TxFIFO** module.

Another important block is the **LineCycle BeaconPeriod Synchronization** one that is in charge of synchronizing the communication by mean of the identification of the **Beacon Period**. As it has been said in chapter 2 when describing the different kinds of **MPDU**, one of those typologies is the Beacon MPDU that is used by the **CCo** to exchange coordination messages with the other **STAs**. The Beacon period is twice the **AC-line cycle period**, so that if the AC-line frequency is 50 Hz, the Beacon period results to be 40 ms. The importance of that value is directly connected with the concept of **Carrier Sense Multiple Access (CSMA)**.
Indeed, before a device (a STA in the AVLN) attempts to transmit data, it "listens" to the power line in order to check if another device is currently occupying the line with its transmission. Then if the medium is busy the device waits for a **random backoff period** before checking another time. This process is used to avoid collisions during the communication. So the Beacon Period is subdivided in equally spaced regions where some of them are dedicated to the transmission on the power line while in the **Stayout regions** there is no possibility for the STAs to communicate information through the medium. The following figure 3.11 taken from [4] gives a graphical representation of the just explained concept.
where it can be denoted the division of the Beacon Period in regions, some of them are dedicated to the transmission on the power line. When it happens, after

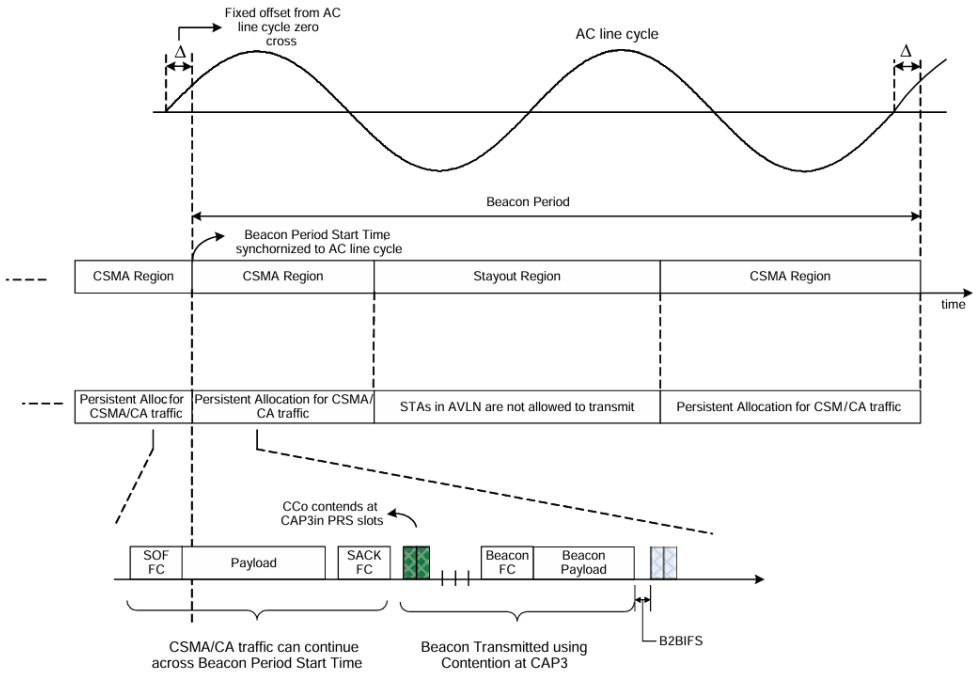

Figure 3.11: Beacon Period structure in CSMA mode (taken from [4])

the receiving of the **Sound acknowledge Frame Control (SACK FC)** that highlights the correct transmission of the device in the medium, there can be seen the two **Primary resolution slots (PRS)** in green which are time slots where the eventual **contentions** of devices that want simultaneously to transmit data are resolved by analyzing the priority level in a scale from **CAP0** to **CAP3** that the CCo has given to the involved stations.
As a matter of fact, in figure 3.9 it is showed that the mentioned **LineCycle BeaconPeriod Synchronization** block and the **CSMA control** one are connected because the latter has to take in input the Beacon Period to solve its function as well as the priority level of the PRS slots.

3.2.4 HPGP Baseband: clock generation unit

The HPGP subsystem includes a **clock generation unit** for providing the required clock frequency to all the different clock domains.
The main clocks are provided to the HPGP clock generation unit by the **HPGP AFE** block derived from the 300 MHz. Configurable clock switches are available to select the required output frequencies while frequency configurations have to be set before starting a PLC communication or between two PLC communication

sessions. The **Clock selection** is controlled by HPGP PHY configuration registers. In figure 3.12 it can be found the scheme of the described architecture.

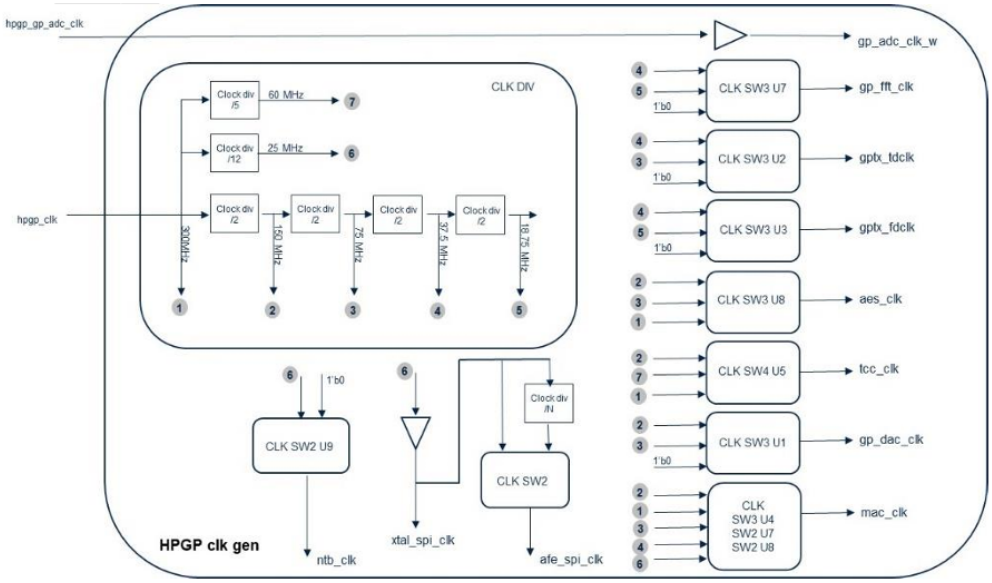

Figure 3.12: Block diagram of the HPGP clock generation unit (taken from [9])

3.3 Digital Kernel

The so called **Digital Kernel** refers to all the digital IPs of the EV PLC Modem project which are highlighted in dark green in figure 3.1. The design of these modules is not a theme inherent to this thesis which is more concentrated on the development of the Monitor IP and the exchanging of data in HPGP format, however in this section it will be given a generic overview about this units, explaining their role inside the whole project. Always in figure 3.1, in light blue are highlighted the **analog IPs** that however will not be treated also because they are very self-explanatory since they basically consist in oscillators, PLL, thermal sensors and a **Power Management Unit (PMU)** which generates all the required voltages for the EV PLC Modem supply, as well as a Power On Reset (PoR) signal which is responsible of releasing the device reset during the power-up.

3.3.1 HPGP AHB and APB ports

The role of the **AHB port** is to read and write data from and to three different **HPGP memory structures**:

- **Ethernet FIFO**

- **SPI FIFO**
- **Packet Buffer**

In particular, the SPI FIFO is not a memory but an input/output port for the communication between the **SPI communication IP** and the **HPGP subsystem**. The IPs that are considered as "masters" of the **AHB SPI FIFO** are the **RISC-V** and the **DMA**, which can access it through a 32-bit access, taking into account that the base address is 0x44010000.
A similar consideration can be made for the **ETH FIFO** which represents an input/output channel for the communication between the **Ethernet communication IP** and the HPGP subsystem. Again, the masters are the RISC-V and the DMA and the access is dof 32 bits starting from an address of 0x44020000.
Something different must be said for the **Packet Buffer** which is a **16KB memory** mapped from 0x4400000 to 0x44003FFF whose access is always held in 32 bits by the same masters of the other 2 memory structures.

The **APB port** is available for programming HPGP registers, indeed in the HPGP subsystem there are two main **register groups** that are available and they are mapped to different addresses:

- **MAC registers** available at 0x44040D00
- **PHY registers** available at 0x44040400

3.3.2 Control Unit (CU)

A **RISC-V** microprocessor represents the **control unit** of the entire Modem IP because it has to control and support the device functionality. It includes a **Local data Memory (LM)**, an **Instruction Cache (IC)**, an interrupt controller and the logic that is needed to support the functional debug features. In addition, a JTAG port is available for controlling the core by an external debugger.
The HPGP functionalities are guaranteed by the core performances, the code is executed either from internal memories **eROM**, **eRAM0**, **eRAM1**, **eRAM2** and **eRAM3** or from an **external Flash memory** which is controlled by an embedded **OCTOSPI** IP that supports standard and quad SPI SDR protocol. In particular, **serial Flash** are mainly used to contain the **EV PLC program** so a decryption on the fly (OTFDEC) could be required for security purpose and it can be capable of decrypting an external program memory from 512 Kbit to 128 Mbit, however the first 8 KB of the external memory space are not decrypted because the flash content has been not encrypted.
Regarding the internal memories it is immediate to notice that there are two bus domains **AHB1** and **AHB2** where AHB1 works at an higher speed with respect

to the other one. This implementation has been adopted because in that way the CU can exploit **two different levels** of memories where eRAM0 and eROM are quicker but smaller memories confronted with eRAM1, eRAM2 and eRAM3 which are slower but store more data. In the following figure 3.13 is reported the external flash path.

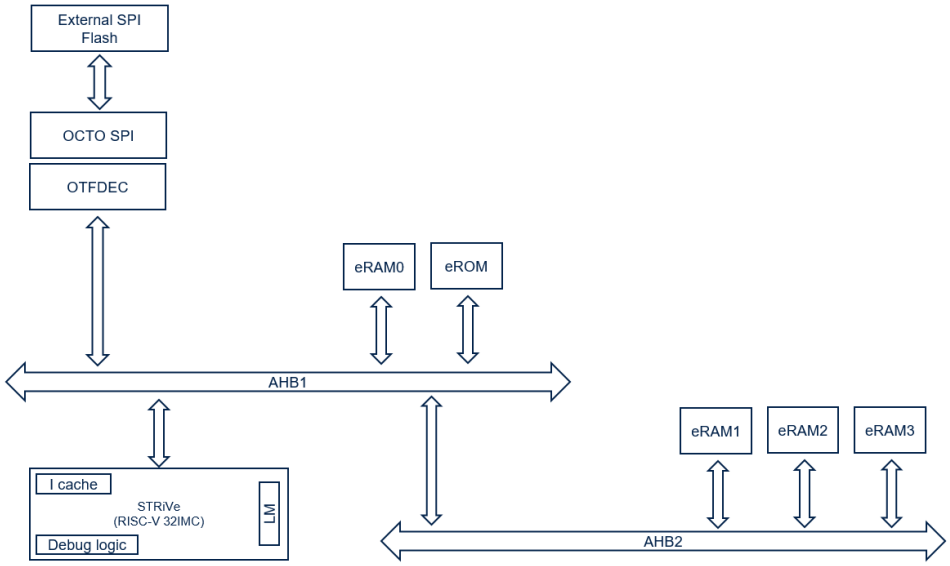

Figure 3.13: External Flash Path

3.3.3 Safety IPs

The concept of **Safety** in the Automotive environment will be deeply explained in chapter 4 as well as the description of the **Monitor IP**, that represents the most important safety IP along with the **Fault Collection and Control Unit (FCCU)** to which is connected through the **hpgp_fail** signal. In particular the FCCU offers a redundant hardware channel to collect errors and to lead in a controlled way to a **safety state** the device whenever a failure is present, without the need of an intervention from the RISC-V.

3.3.4 Security IPs

The difference between the concepts of **Safety** and **Security** in Automotive will be also explained in chapter 4, however while the safety regards the recovery from a **physical failure** of the system, the latter is about the prevention of an eventual **external violation** that can be held to tamper the device.

In the EV PLC Modem context, the security IPs are required for supporting **encryption** and **decryption** functions. These features will be mainly used for authorizing the executed code and for implementing security on the data transfer by the communication IPs. All the required features are implemented in the HW secure modules:

- **HASH**, which implements an **HASH algorithm** that takes an input (message) and outputs a string of characters which represents uniquely the input. Hashes are used for verifying data integrity.
- **Advanced Encryption Standard (AES)** which is a symmetric encryption algorithm that works with keys of varible length.
- **Public Key Accelerator (PKA)**, an HW module that is needed for accelerating the encoding operations that involve public keys.
- **True Random Number Generator (TRNG)** that generates random numbers based on unpredictable physical phenomena as the **thermal noise**.
- **On The Fly Decryption (OTFDEC)** that has been already introduced before and it is a module which allows the data decryption in real time.

All these modules work together in order to improve the security of the system and preserve its integrity.

3.3.5 Communication IPs

In the following figure 3.14 there is a block diagram of the EV PLC Modem which highlights all the external connections with the latter.
As it can be seen, the EV PLC supports **Ethernet** and **SPI** protocols for data exchange with the host processor. For the supporting of the Ethernet communication, it has been inserted a **ETH GMAC** IP inside the Modem architecture while the **ETH PHY** must be added externally and controlled by mean of a **Management Data Inpput/Output (MDIO)** interface which is a serial bus defined for the IEEE 802.3 Ethernet standard that is needed for connecting an Ethernet MAC with its external Ethernet PHY while the data communication between the EV PLC and the ETH PHY is held by mean of the **RGMII/RMII** interfaces.
In addition, a **SPI slave configuration** has been added for the supporting of the **SPI communication**. Moreover, an **SPI request line** which is part of an SPI driver has been added to the standard SPI protocol for giving the possibility to the EV PLC Modem to start the communication.

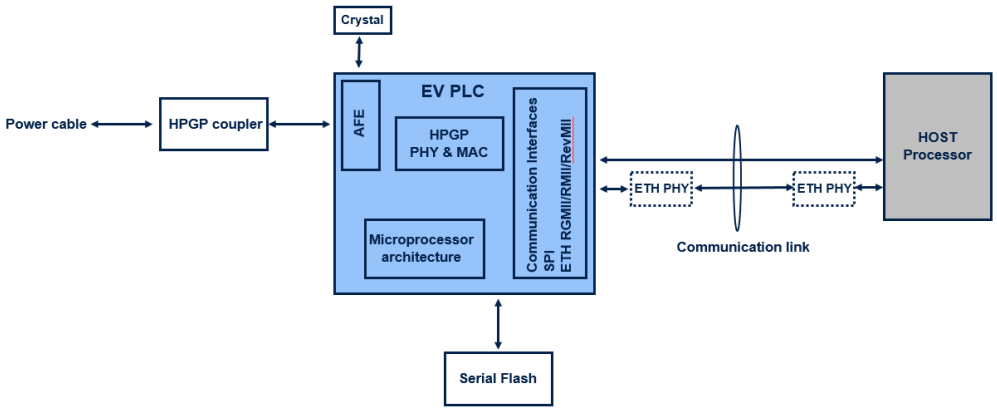

Figure 3.14: EV PLC application block diagram (taken from [9])

3.4 EV PLC system configurations

In this section will be presented the several configurations of the EV PLC Modem system. As a matter of facts, it can interact with the **host processor** basically in three ways that have also been partially introduced when talking about the communication IPs.

The system can be connected and interact with the host processor by mean of its SPI interface and the use of an external serial flash as it can be denoted from figure 3.15.

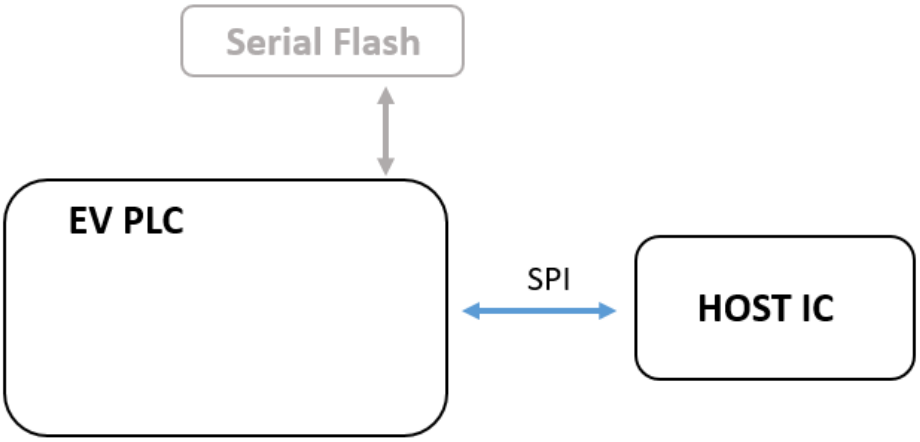

Figure 3.15: EV PLC SPI configuration

As an alternative it is possible to have an **Ethernet connection** with that host processor if it is required by the bandwidth or by the system topology, however it can be held by mean of an **external ETH PHY module** or by mean of an **RGMII/RMII digital connection** if the topology allows for a point-to-point connection between the Modem and the host processor or at least a very short distance. In this last case the advantage is the BOM cost reduction. The first case is presented in figure 3.16 while the latter in figure 3.17.

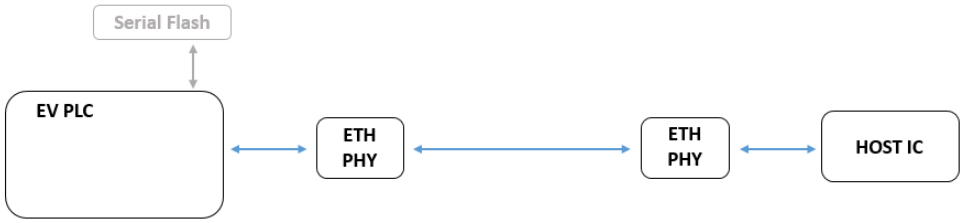

Figure 3.16: ETH with external PHY

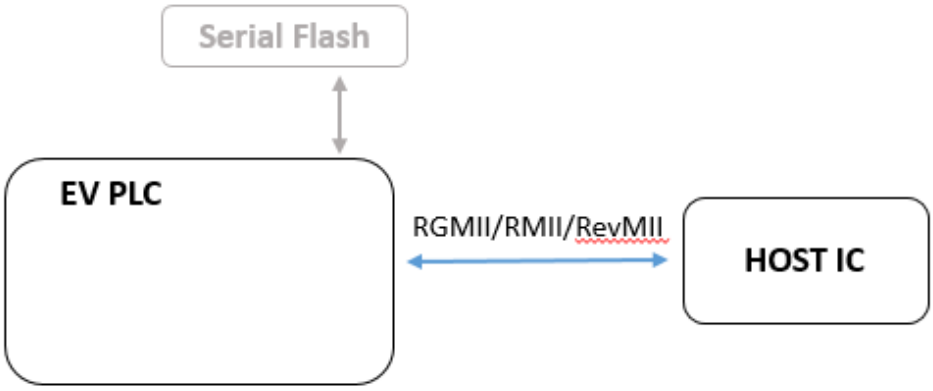

Figure 3.17: ETH without external PHY

Another aspect that must be analysed is the way the **program code** is get and executed by the RISC-V of the Modem. Also in this case there must be done a differentiation based on the different possible two configurations. One of them consists in the storage of the boot code in an **external flash memory** whose dimensions can go from 512 Kbit to 256 Mbit. This is the case that require the less interaction with the host processor. The code stored in the flash can be encrypted with the correspondent decryption executed runtime by the OTFDEC. This configuration is represented in figure 3.18.

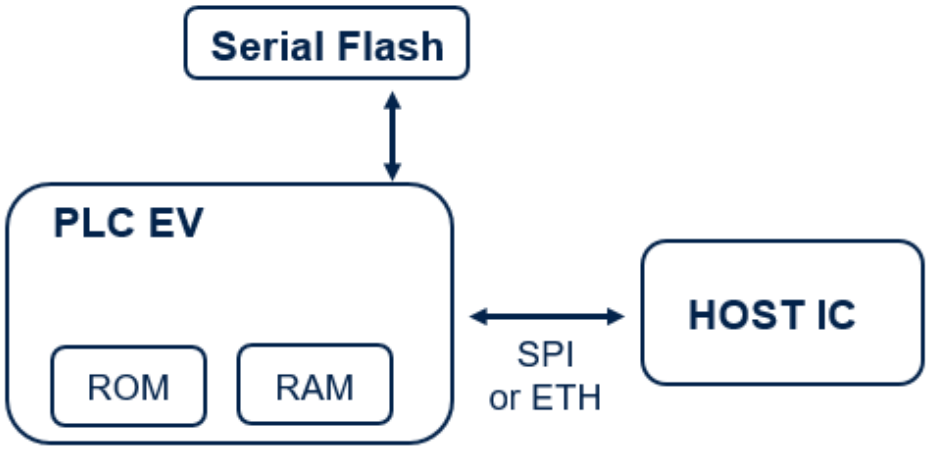

Figure 3.18: EV PLC with external flash memory

The other possibility is to download the program code from the host processor to the internal memories by the communication IPs **Ethernet** and **SPI**. In this case the total boot time can be higher than the previous case because the host processor has to complete its boot before supporting the EV PLC one, furthermore the program code size is limited by the available embedded memory but the BOM is optimized since this time an external flash is not required.

In general the **boot modes** are driven by the so called **bootsel pins** on the **IO ring** of the Modem. The link between the pin values and the configurations is reporting in table 3.1:

BOOTMODE	BOOTSEL1	BOOTSEL0	BOOTLOADER BEHAVIOR
Test	1	1	The Bootloader executes the self-testing code, provides the result and then enters an endless loop
Execution (normal)	0	0	The Bootloader loads and executes the code read from an external NVM device attached to the OCTOSPI interface. If it does not find any valid boot code it enters an endless loop
Boot from Ethernet	1	0	The Bootloader endlessly waits for valid boot data from the Ethernet interface
Boot from SPI	0	1	The Bootloader endlessly waits for valid boot data from the SPI slave interface

Table 3.1: Boot code configurations

Chapter 4

HomePlug Green PHY Monitor

4.1 Introduction to the Monitor IP and its process of realization

This chapter will present the development process of the **HPGP Monitor** IP of the **EV PLC modem**.

In particular this is the last IP that has been chosen to be added to the project, this because the monitoring of the most relevant signals is of strong importance, especially in the **automotive** field, where it is fundamental to sense immediately the possible failures in order to recover from them or to take decisions that potentially can save the life of the user.

Since in this case the topic is related to the charging of an EV and especially to the exchanging of information between the latter and its EVSE, the failures are represented by errors in the **communication frame** that has been deeply explained in chapter 2. Such errors may happen because of different kinds of problem during the communication, but maybe the most relevant of them is the noise that may be introduced in the **receiving chain** of the **HPGP subsystem** at the level of the data that are exchanged on the **power line** and so received by the modem in receiving mode at the level of the AFE.

The development of this IP represents my practical activity inside the **ADAS and Infotainment for automotive** team of ST and it started with an accurate study of the **HPGP subsystem** provided to ST by the already mentioned external company

and analyzed in chapter 3. As a matter of facts, as a first thing it was important to have a complete comprehension of the architecture, comparing it to the concepts explained in the **HPGP standard** treated in chapter 2, this for having a precise idea of the HPGP signals to monitor in order to implement a meaningful monitoring function and identify precisely the errors that may be expected. The choice of the signals and the study of the functioning of the HPGP subsystem was also achieved by the direct observation of its behavior by mean of a proper **simulation environment** that has been set by our ST team. After these steps, the IP has been practically realized in **VHDL** and **System Verilog** and then simulated to verify its correctness. As a final step the developed architecture has been verified, synthesized and then included it in a new **bitstream** to be loaded into an FPGA that is used in the laboratory in order to emulate the functionalities of the whole modem.

The standard **project flow** that is usually followed in the company when developing a new IP starts with the **RTL implementation** of the architecture that than is verified through the use of a **Software simulation** tool in order to check if the function of the implemented IP is the expected one otherwise all the problems are corrected and the IP again verified. If the verified RTL implementation gives good results, the next step is to produce a **bitstream** of the architecture to be loaded in a **FPGA prototype** that emulates the real system in the lab. This step is very important because it allows to test the functionalities of the developed IP in conditions that are closer to a real implementation. This allows the **tuning of parameters** on the basis of a real use case and not a software simulated one that would take much more time for having a test with the same completeness. When the tuning phase is completed, the silicon chip can be realized and so an **ASIC** which implements the entire IP is produced. However sometimes there is the need of tuning again some of the parameters since the real and final implementation of the system in the ASIC may have different features and conditions with respect to the prototype. There is also the possibility that in this final phase of **Chip Validation** some structural problems are discovered. In this case the only possibility is to come back to the RTL implementation and start again all the workflow in order to release a better final ASIC. Of course this situation has to be avoided because it leads to a delay in the project and a waste of resources for the company, this is why there are many verification steps before arriving to the final implementation of the IP. This process is depicted in figure 4.1.

In the case of this project, since the Monitor IP has been decided to be added in a final phase close to the **Tape Out** of the final chip, the one presented in this thesis is a **first version** of the Monitor where our team only had the opportunity to observe a basic implementation in the FPGA protoype with the entire Modem system and refine some of the most important parameters. With an high probability

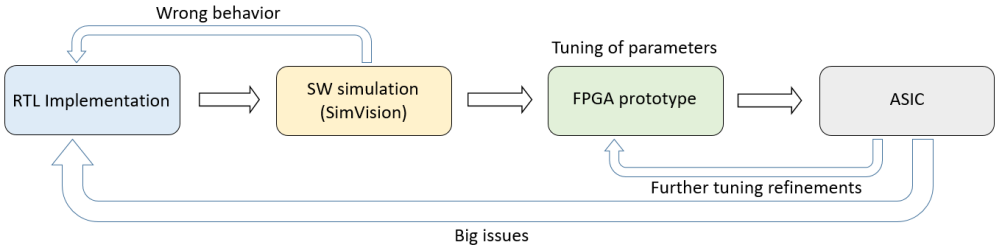

Figure 4.1: Standard IP workflow followed in the team

the Monitor IP will be extended and improved in the next months after a first version of the ASIC. For this reason at the end of this chapter there is a dedicated section 4.5.5 that reports some of the possible future improvements than can be applied to the Monitor IP.

4.2 Why is there the need of a Monitor IP?

4.2.1 HPGP Monitor features

At this point a reader may ask why there is the reason to develop a dedicated IP for monitoring some signals that are taken from a functioning IP and that are already indicating some error conditions by themselves.

As a matter of facts some of this signals that are taken from the **HPGP Subsystem** provided to ST by a specialized company and used in the Monitor IP have been originally created to indicate error conditions in the received frames. However the study of the HPGP frame that has been addressed in chapter 2 has taught that a communication frame is composed by three parts: **preamble**, **frame control** and **payload**. Given the differences between them it was reasonable to imagine that not all of the errors may have the same effect towards the system, for instance the communication may not be considered corrupted if there are some frames that have errors in their payload portion that is the one carrying data while just a single frame that has a frame control error which makes the frame bits containing the information on the ID of the receiving station to be incorrect can make the entire communication to be wrong.
For this reason it was necessary to think about a solution that can identify this errors in the received frames using those signals coming from the HPGP subsystem but that can also be able to **count** the number of the received frames as well as the number of identified errors, categorized on the basis of which part of the frame they involve. Then this IP should consider some specific **thresholds** for each kind

of error whose exceeding lead to a **fail condition** that has to be managed by the **FCCU** which at its turn may decide to reset the whole modem in order to recover from the error.

Moreover the Monitor IP was thought also for making some signals to be "visible" to the RISC-V in order to be read and analyzed via software. As a matter of facts the HPGP subsystem has a group of **debug signals** that are inherent to several intermediate results given by the digital blocks which compose the HPGP subsystem. Since the Monitor IP is provided with a **register interface**, these signals can be acquired and sent to the RISC-V by mean of the AMBA APB. It can be important to have access to these signals because if something unexpected goes wrong or for some reason the team has the necessity to observe a particular internal value of the HPGP subsystem, it will be possible to access those values using the Monitor by reading its registers.

4.2.2 Functional Safety

It has to be said that the HPGP Monitor has been introduced to increase the level of **Functional Safety** of the Modem, which is a very important theme in the Automotive field.
Functional Safety refers to the handling of risks that derives from some **system failures** (both HW and SW) that may happen in a developed device which can damage even seriously and permanently the user. To address the Functional Safety means making the system to behave in a predictable and controlled way even if it is in a fault condition.
The **ISO 26262** [10] is the most authoritative guideline for the companies that produces Automotive devices for facing the Functional safety issue. As a matter of fact the aim of that standard is to address the different kinds of hazards regarding the malfunctioning behaviour of electronic systems in EVs. The main goals of the ISO 26262 are to establish the guidelines to adopt in order to **identify the possible failures** that may happen in the developed device as well as methods for developing the system being sure that all the Safety related problems are being addressed. It also classifies the risk for an Automotive IP subdivided in 6 categories named **Automotive Safety Integrity Levels (ASILs)**: QM, ASIL A, ASIL B, ASIL C and ASIL D where the first one indicates an application with no automotive hazards while the others are related to an increasing degree of hazard.

In the case of the EV PLC Modem, it has been realized in such a way that it can be used in applications targeting up to **ASIL A** meaning that since there are not potential high risks, less strict design procedures have been used during the

development. However a lot of safety related mechanisms have been introduced in the Modem such as a **PMU Voltage Monitor**, a **Clock Monitor Unit** and a **Temperature Monitor** as well as the **FCCU** whose functionalities are related to the treated Monitor IP that so improves the Functional Safety of the device.

One key point of the ISO 26262 is the classification of **failures** which is a term that is defined in the standard as the cessation of an expected behavior of an element or element due to a manifestation of failure where the cessation can be permanent or transitory. As a matter of fact defining the several kinds of failures is strongly useful in order to take decisions regarding the realization of a **failure recovery mechanism**. The classification of the various failure types is done when dealing with the **Hardware level** of the project. In the standard, the classified **failure modes** are the following:

- **Safe fault (S)**: a failure that if occurs it does not increase the probability of violating a safety objective.
- **Single-point fault (SPF)**: an hardware failure that leads directly to the violation of a safety objective and it is not recovered by any kind of failure recovery mechanism.
- **Multiple-point Fault (MPF)**: a single hardware failure that, combined with other failures, leads to a failure in more points of the system.

Apart from the definitions of the different types of failures, in the standard ISO26262 there are also the definitions of the **Hardware Architectural Metrics** that are needed in order to evaluate the efficiency of the **hardware architecture** in terms of safety. These are essentially 2 metrics:

- **Single-Point Failure Metric (SPFM)**: it is a parameter that indicates the robustness of an architecture towards single-point failures.
- **Latent Failure Metric (LFM)**: it indicates the robustness of an architecture towards failures that are latent in time.

In the documentation of the EV PLC Modem, for every safety IP or function that has been implemented it is specified if that feature is SPFM or LFM robust or both in order to check if the ASIL A requirements are satisfied. In the case of the Monitor IP that has been developed, since it is one of the most important safety IPs of the entire system along with the FCCU to which it is connected, it has to be resilient to both single-point and latent failures.

Another concept regarding the safety topic that must be explained is its difference with respect to another very important requirement for the automotive

field IPs which is the **security**. As a matter of fact this two requirements are often confused but they are intrinsically different. The security in an automotive system regards all the features that are implemented in order to make it invulnerable to possible attempts to tamper with the system itself. These "external attacks" could have the purpose to steal data from the system or to corrupt it in order to provoke damage to the users. Actually it is more often referred with the term **cybersecurity** since a large part of the possible attacks can be made via software. So having a look at the previous definition of security it can be noticed that it is different from the safety definition that is oriented to the protection of the users towards accidental (and not provoked) failures that may happen basically for **physical damages** of the system.

4.3 An high-level overview of the Monitor IP

The first step in the Monitor IP realization has been to think and draw a generic but powerful scheme of the architecture in order to establish the main actors involved in the process. The following figure 4.2 represents the most high level representation of the Monitor IP architecture:

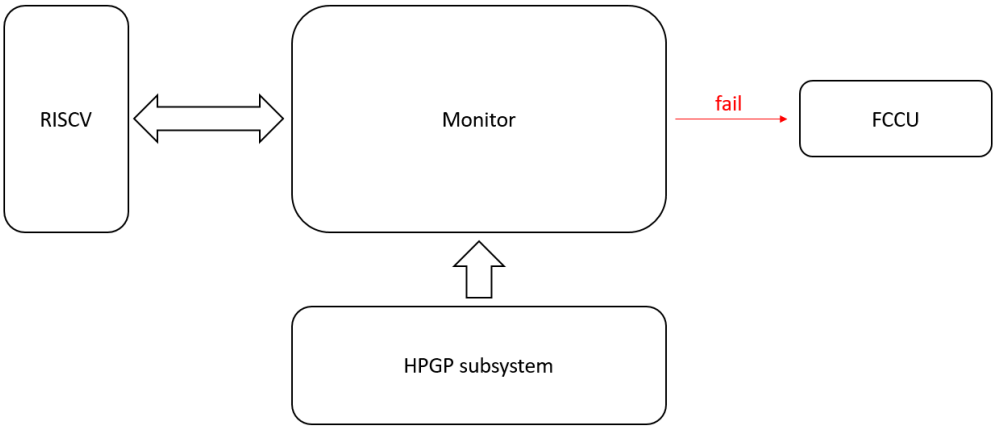

Figure 4.2: High level representation of the monitor architecture

The first thing that has been considered while designing the monitor is that it must be a digital architecture that outputs a single bit **fail** signal. This because as briefly introduced before, the monitor is the latest IP that has been introduced in the design of the modem, so it must has been introduced in such a way to be consistent with the already present IPs. Since the **FCCU** is a standard model from the ST library and has been taken as it is, the output of the monitor should be just of one bit in order to be compliant with FCCU entity.

The top left block in the figure is the **RISC-V** processor. As described in chapter 3 it plays an important role for all the digital IPs in the Modem because it is responsible of their configuration through the use of the **AMBA bus**. Also in the case of the Monitor IP it receives commands from a Microprocessor that is programmed using the C language to set the registers of the Monitor that store its configuration signals. These signals that will be treated later in the thesis go from **enabling** signals to the setting of some **error thresholds** whose exceeding produces the **Monitor fail signal**. The registers are instantiated in a proper block named **APB interface** that is internal to the Monitor and that will be addressed in the next sections.

Another observation that must be done is that there must be a double-direction connection between the RISC-V microprocessor and the monitor because the goal is that not only some control signals must be sent to the monitor by the processor but also the signals coming from the HPGP subsystem need to be seen by the RISCV so that they can be used via software as the **debug signals** that have been briefly introduced before.

The other main block is the **HPGP subsystem** that is the main IP of the modem from which the most important signals to be monitored are extracted. As it was briefly mentioned in the introduction of this chapter, a large part of the time has been spent in order to prepare a proper **simulation environment** for observing the most meaningful signals which highlight the possible presence of different kinds of error that can occur in the HPGP communication frame. In section 4.4 will be explained how this environment has been prepared and which are the designated signals of the HPGP subsystem that have been exploited to count the number of wrong frames and make the fail signal to rise.

4.4 Simulation environment and monitored signals

4.4.1 Frame issues to be faced

At this initial phase of the development of the Monitor IP the priority was to build a simulation that was necessary to study the various signals produced by the **HPGP Subsystem** IP of the Modem in order to highlight a frame whose content is wrong because it has been **corrupted**. So it is obvious that a preliminary operation must be of course to identify which factors make a frame to be corrupted.

In every communication system, a first problem to address when exchanging information is the **attenuation** given by the medium that in this case is the power line. As a matter of facts it can happen that despite a Modem is transmitting a frame properly, the latter can be strongly attenuated before arriving to the receiving Modem so that this one is not able to recognize it due to the **low energy** level of the frame. It has to be said that a sort of attenuation is always present whenever a frame is transmitted across the power line. This is the reason why, as reported in chapter 3 when dealing with the RX chain of the Modem, the AFE presents an **Voltage Gain Amplifier (VGA)** whose gain can be set by some control signals of the Baseband. This amplifier is useful to set the signal received to a certain dynamic in order to be correctly demodulated. However the problem occurs when the level of attenuation is such low that even the amplifier is not able to apply a sufficient gain for taking the received frame to a correct dynamic level. This is a case of frame corruption in which the information is not sensed by the receiver.

The other main problem that can corrupt the frames and make the communication invalid is **noise**. The effect of noise in digital communications is a very big topic to deal with because it may depends on many factors such as the context on which the communication system is applied as well the different kinds of noise that can disturb the system. In the context of the EV PLC Modem the main categories of noise that can disturb the communication are:

- **Impulsive Noise**: the turning on/off of nearby devices may cause a narrow high amplitude noise which can degradate the quality if the communication.
- **Narrow Band and Wideband Noise**: the narrow one can be caused by devices that work at specific frequencies such as wireless communication devices while the wideband one is more correlated to power supplies that are not shielded in a proper way and can potentially reduce the available bandwidth for the communication.
- **Periodic Noise**: it is the case of a noise that repeats periodically with the frequency of the electricity grid which typically is produced by a device that operates in synchronicity with this 50/60 Hz frequency like an household appliance. These periodic disturbances may affect the **transmitted beacon** and so the synchronicity of all the stations in the HPGP network.
- **White Noise**: it is caused by thermal phenomena and can decrease the quality of the communication.
- **Electromagnetic interference (EMI)**: it is due to an external electromagnetic field which interferes with the power line introducing into them unwanted signals.

After having analyzed the main problems that can occur during a communication, it is important to apply these considerations on the specific case of the project where a huge number of frames is transmitted in a time window. Regarding the **attenuation** it is obvious to imagine that it is uniformly applied to all the frames that are transmitted during the communication or at least to a large group of them, this because of the causes of the attenuation itself that are mainly due to the **length** and the **topology** of the power line that so regard all the signals involved in the communication. Even more so there is no reason to consider that part of a single frame can be attenuated and another part not, so for instance it is not possible that a frame presents a **preamble** that is strongly attenuated and a **payload** with a smaller attenuation. Then, as said previously, what can be expected is that under a certain level of attenuation the Modem is no more able to tune the VGA to sense the received frames.
For the **noise** issue, since it can be applied to the transmitted frames in different shapes and with different characteristics, there is the possibility that a noise impulse could be applied just on one part of a single frame (even if it is very difficult), so there is the need to make a distinction between a **preamble error**, a **frame control error** and a **payload error**. The way the HPGP subsystem is able to recognize and distinguish the different kinds of errors in the frame will be discussed in the following sections.

4.4.2 Overview on the simulation environment

Before going into the details on how the HPGP subsystem can sense the different errors that may occur in a frame and which are its signals that make these errors to be monitored, it worth the effort to explain how a proper simulation environment has been set up in order to make such considerations. The following figure 4.3 represents a schematic useful for the understanding. As it can be seen, in a real case the HPGP subsystem of a RX Modem should receive 12-bit samples from its AFE. This because the frame is transmitted by a TX Modem in the analog domain using the power line as a medium, then this analog signal is sampled by the **12-bit ADC** in the AFE of the RX Modem. However since the actual goal was to simulate the functionalities of the HPGP subsystem in order to find some signal of the Baseband that can give information about the correctness of the received frames, an important operation to be done was to **create different types of wrong frames** and load their samples directly in the baseband **bypassing the AFE**. This because it would be very complicated to make a TX Modem to produce some wrong frames in a controlled manner, moreover the simulation would have taken much more time than in this case where the simulation instantiates just one Modem, the one in RX mode, and the transmission of the frames is under

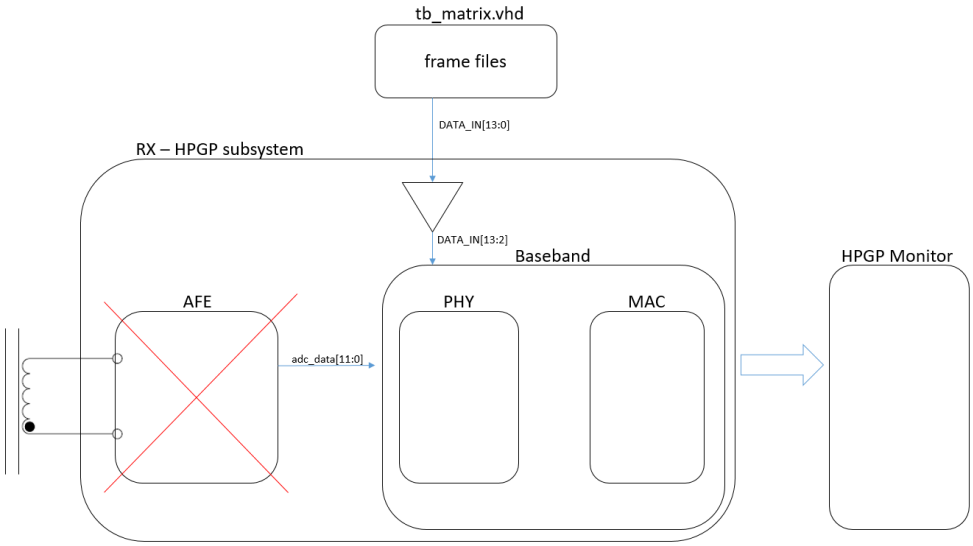

Figure 4.3: Schematic of the simulation environment

the responsibility of the **tb_matrix.vhd** block that is in charge of **reading the 11-bit samples** of the frames from their text files to provide them as input data of the PHY block in the RX Baseband.

It has to be denoted that the informative content of the samples is on 11 bits but both the **DAC** of the TX chain and the **ADC** of the RX chain work on 12 bits so the 11 bits have been padded with one 0 at the LSB. This also makes the samples more robust towards noise. Moreover the output of the tb_matrix block is on 14 bits since the output signal is **DATA_IN[13:0]**. The reason is that this block has been derived by another project where it was needed an output of that dimensions. In this case the choice has been to maintain the entity of the block but padding the samples read from the files to reach 14 bits. Than as it is represented in figure 4.3, the 2 LSBs of the DATA_IN signal have been discarded in order to get the 12-bit input as it should be if there was the standard AFE that samples the transmitted analog signal.

Once the simulation environment has been set, the attention shifted to artificially create using **Matlab** some frames with different kind of errors in order to have the possibility to observe the "reaction" of the HPGP subsystem to these errors in such a way to be able to send the most meaningful signals to the Monitor IP.

4.4.3 Frames dumping

Now that the simulation environment has been exposed, the next step of the explanation regarding the project flow of the Monitor IP has to be the **generation of the test frames**. As it has been introduced earlier, the purpose is to create some frames that are based on the samples of the original correct test frames that has been provided to ST Microelectronics by the external company that developed the HPGP subsystem, but some of these samples have to be changed in order to simulate a possible noise that can corrupt the frames or part of them during the transmission.

The first thing that has been done, was the **dumping** into text files of the samples of the original correct frames from the simulation environment provided to ST Microelectronics. That simulation environment is different from the one described in the previous section, indeed this one takes into consideration two EV PLC Modems where one is set in transmission mode and the other acts as the receiver as it is represented in figure 4.4 that highlights the interaction and the exchanging of data between the two main entities.

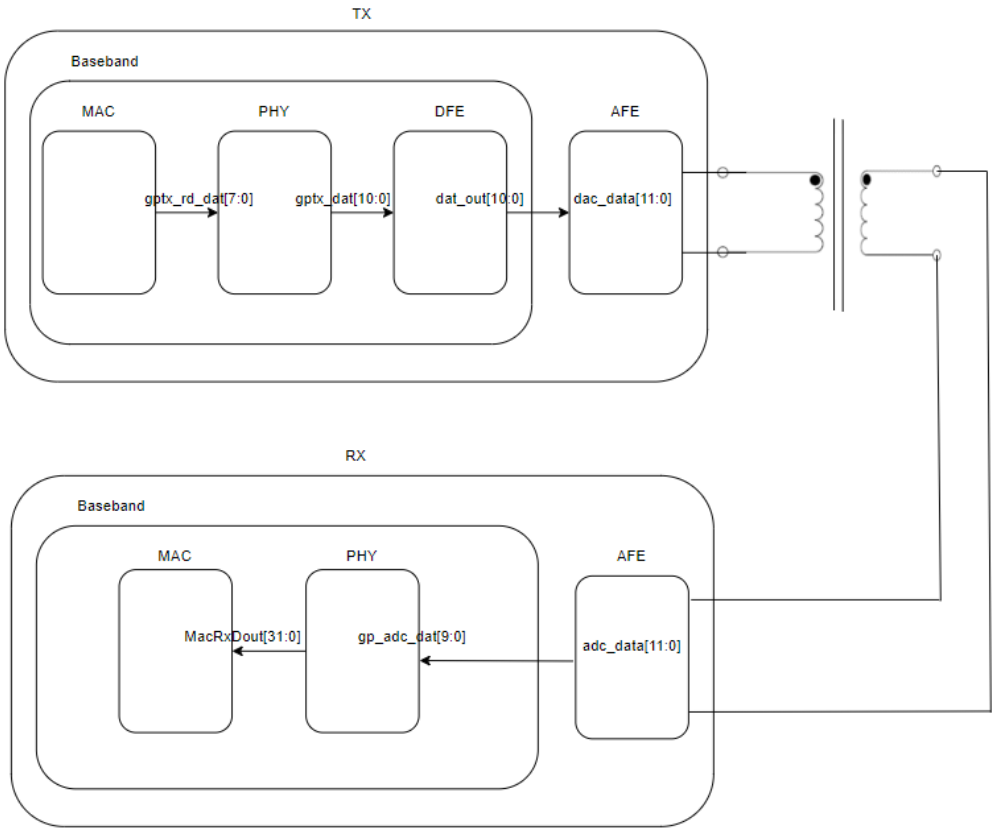

Figure 4.4: Original simulation Environment exploited for the frame dumping

Looking at that figure, one may be dumbfounded by the changing on the dimensions of the data signal from the transmitting device to the receiving one. That is because it is a strongly simplified schematic that just has to give the idea of the main blocks involved in the simulation and the most important exchanged data signals. These blocks are at their turn composed by a multitude of other blocks that implement many functions and change the dimensions of the data signal, the majority of these blocks have been analyzed in chapter 2, especially the ones regarding the PHY block that is the one in charge of modulating and demodulating the data. However, from figure 4.4 it can be seen that the output signal of the PHY block is **gptx_dat[10:0]**. This signal enters the **DFE** block that is in charge of interpolating it in order to double its sampling frequency and whose output is padded with one zero on the LSB and represents the input of the **DAC** on the TX AFE. It is easy to understand that the informative content of the transmission is contained in that 11-bit signal that comes out from the PHY block and is the result of the **modulation** of the MAC data. The streams of the values assumed by that signal represent the **samples** of the transmitted frames, or to be precise, the 11 MSBs of the samples (the LSB is always zero). This for saying that for sure this is the correct value for making the dump of the frame and obtain the text files with the samples. Figure 4.5 presents the 4 different frames that have been used in this simulation that has been provided and the streams of the values of the signal **gptx_dat[10:0]**.

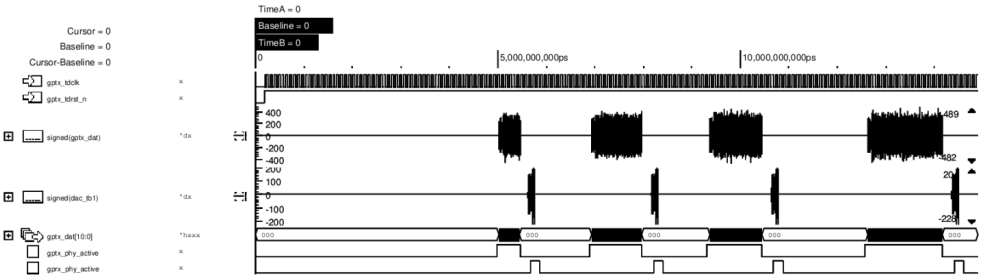

Figure 4.5: Frames to be dumped and correspondent samples

In the previous figure, apart from the clock and reset signals, one can observe the **sampled representation** of the 4 different transmitted frames and their correspondent **acknowledgement** signals. As a matter of fact once a frame is transmitted, if the transmission has been done without problems, the receiver sends back to the transmitter an acknowledgement signal which communicates some information about the good outcome of the communication. Since these signals have been taken from the **transmitter**, the **gptx_phy_active** signal, which indicates the transmission time window, is high when the frame is transmitted and the analogous **gprx_phy_active** signal is asserted when the acknowledge is received from the transmitting EV PLC Modem. If those "time window" signals

would have been taken from the receiving IP they would be reversed as the receiver Modem is the one that receives the frames and transmits the acknowledgments signals.

Regarding the frames themselves, they are composed by a variable number of samples. From figure 4.6 the three parts of the frame introduced before and deeply analyzed in chapter 2 can be distinguished.

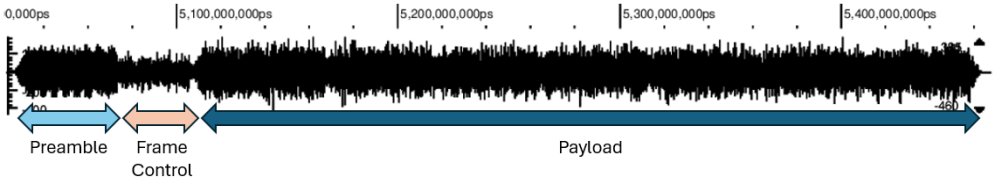

Figure 4.6: Analysis of one of the transmitted frames

The subdivision of the frame in its composing parts as proposed in image 4.6 has been done both by knowing the theory behind the frame composition and empirically by looking at the sampled representation of all the frames. This because from the theory one can get that the three parts follow one another starting from the **preamble** for than going to the **frame control** and the **payload** and by looking at the frames the three parts can be distinguished by eye. Moreover it is known that the preamble is always a single symbol defined in a given time window so it has always the same number of samples that have to be the same for every frame, since it is a mathematical value that is inserted in the TX chain of the PHY block after the modulation of the information to be transmitted with the purposes of **synchronization** and **estimation of the energy** for the receiver. The external company that projected the HPGP subsystem provide to the ST team an Excel file with the exact 3456 samples values for the preamble. Regarding the frame control and the payload the theory on the HPGP standard says that, unlike the preamble, they can vary from frame to frame, indeed there can be frames with 1 or 2 symbols of frame control and a variable number of symbols for the payload (there can be also no payload). In the case of this simulation, the 4 frames are just distinguished by the different length of the payload that is the portion of frame carrying the actual information to be transported.

At this point knowing that the frames have to be sampled by the ADC at 75 MHz, what have been done was to **dump** the values of the samples contained in the gptx_dat signal which contains the data of the frames sampled every 13.344 ns which corresponds to a sampling frequency of 74.94 MHz due to a slight mismatch of the clock in the simulation with respect to the theoretical one. In figure 4.7 is represented a portion of one frame with a zoom on some of the sampled values.

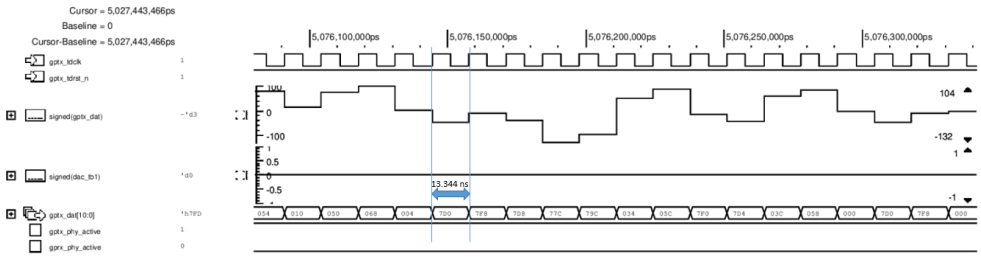

Figure 4.7: Zoomed portion of one frame

4.4.4 Wrong test frames generation

Once the text files containing the frame samples of the four frames have been derived, the attention shifted towards the **introduction of errors** in them in order to get frames with different errors for being tested in the developed simulation environment and so observe the behavior of the receiving HPGP subsystem signals.

In order to do that, a **Matlab code** has been developed. In the first part of it the most important parameters such as the bits of the **DAC of the transmitter** that converts the digital samples in the analog frame to be transmitted on the power line, the sampling period and the sampling frequency are defined. Then the text files are opened and the hexadecimal samples are read as integer numbers. However the Matlab function converts the hexadecimal values in integer numbers as they are not **signed** values. So in order to consider the sign of the hexadecimal samples, the quantity 2^{11} is subtracted from the integer quantity if it is greater than the half of the maximum range of the DAC. Furthermore in order to represent more accurately the frames in Matlab, the range of representation of the frames must be extended because actually they are represented in the range:

$$[-2^{11-1} \div 2^{11-1} - 1] = [-1024 \div 1023]$$

but that would be better to shift to the more accurate range of:

$$[-2^{16-1} \div 2^{16-1} - 1] = [-32768 \div 32767]$$

so having a representation of the samples of 16 bits. For obtaining this increasing of the samples accuracy it is sufficient to multiply every frame for the quantity $2^{(16-11)}$. The above passages are the explanation of the following code where **MSDU1.txt** indicates the text files containing the samples of the first frame represented in figure 4.5.

```
input_dir = 'D:\DOCUMENTI\POLITECNICO\TESI\matlab\
   ORIG_FRAMES\';
filename_in1 ='MSDU1.txt';

dacBits = 11;
Ts = 13.344*1e-9;
Fs = 1/Ts;

% Reading and conversion of hexadecimal data for MSDU1
fin = fopen([input_dir filename_in1], 'r');
I = int16(fscanf(fin, '%x')); % Read data as hex
fclose(fin);

idx = I >= 2^(dacBits-1); %2^(dacBits) is half of the
   max range for the DAC
I(idx) = I(idx) - 2^(dacBits);
I = I * 2^(16-dacBits);
```

In the following figure 4.8 there can be appreciated the Matlab plot of the first frame in the time window in which it has defined. The amplitude indicates the integer number value of the samples.

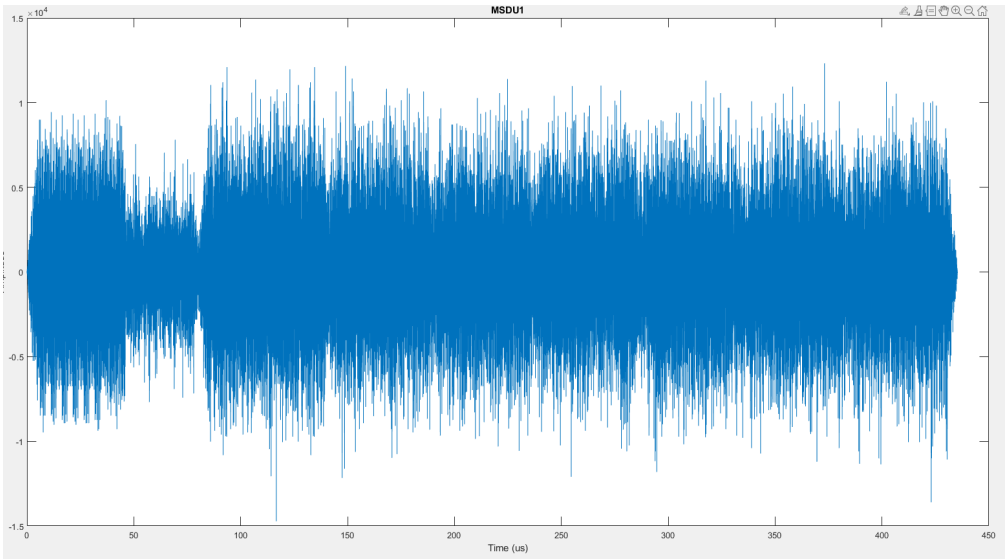

Figure 4.8: Matlab time representation of the first frame

It is interesting also to have a look at the **Fast Fourier Transform** plot in figure 4.9 of the frame because it can be noticed that the relevant content of the frame

is carried approximately in the frequency range $2 \div 28$ MHz as it is stated by the specifications in the HomePlug Green PHY standard.

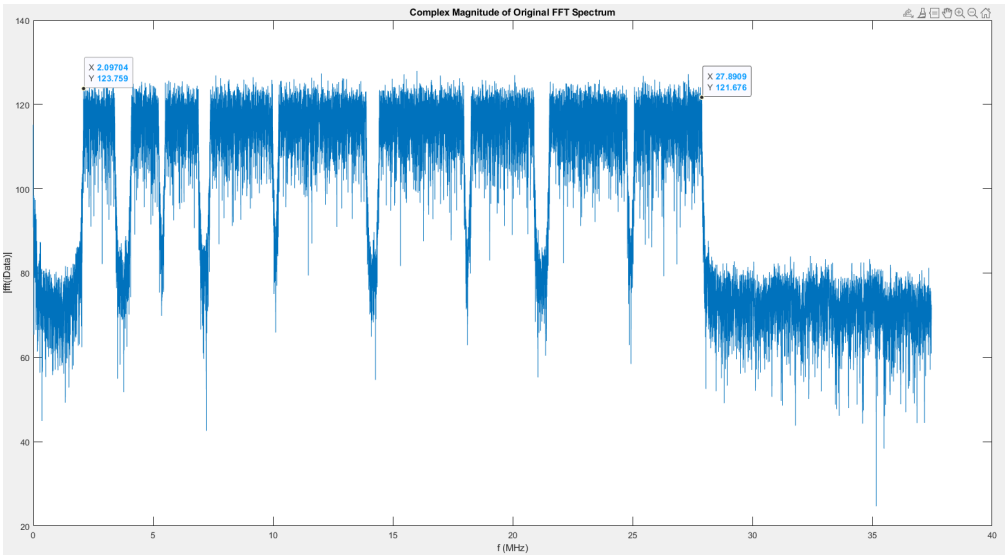

Figure 4.9: Matlab frequency representation of the first frame

Proceeding with the overview on the Matlab code, the first kind of frame corruption that have been applied is the **attenuation**. In particular, two different degrees of attenuation have been applied to the original frame, one at **6 dB** and the other at **40 dB**. The purpose of having two frames attenuated at different levels was to observe possibly a different behavior in the response of the signals of the HPGP subsystem. As it has been said previously, the expectation was to have a certain level of attenuation such that the VGA of the receiving AFE was no more able to increase the gain to set the input signal to the desired dynamic range. In that case the input signal, if strongly attenuated, should not be "seen" by the receiver for its low energy. In order to apply the attenuation levels to the original frame samples, a simple Matlab function has been developed:

```
1 function [frame_out, fft_frame_out] = attenuation(
    frame_in, att_db)
2     fft_frame = fft(frame_in);
3     fft_frame_out = fft_frame .* (10.^(att_db/20));
4     frame_out = round(real(ifft(fft_frame_out)));
5 end
```

which simply takes as inputs the sample values along with a desired dB value of attenuation, then the Fast Fourier Transform of the samples is multiplied by the

linear value of attenuation and the output is computed by making the Inverse Fast Fourier Transform and a rounding of its result. **frame_out** represents an array of attenuated samples of type double that is long as the number of samples of the frame that in the case of MSDU1 is of 32627 samples. The function is applied to the main code and the results are printed to other two different files:

```
%%% attenuation %%%
[MSDU_att_6dB, fft_MSDU_att_6dB] = attenuation(I, -6); % x dB attenuation
[MSDU_att_40dB, fft_MSDU_att_40dB] = attenuation(I, -40);

% - - -

output_dir = 'D:\DOCUMENTI\POLITECNICO\TESI\matlab\TXT_outputs\';
filename_out = 'MSDU1_att_6dB.txt';
pathOut = [output_dir filename_out];
printToFile(pathOut, dacBits, MSDU_att_6dB);

output_dir = 'D:\DOCUMENTI\POLITECNICO\TESI\matlab\TXT_outputs\';
filename_out = 'MSDU1_att_40dB.txt';
pathOut = [output_dir filename_out];
printToFile(pathOut, dacBits, MSDU_att_40dB);
```

by mean of a simple **printToFile** function that takes the attenuated samples of type double, formats them again in the original 11 bits range of the DAC and prints them in a specified text files in hexadecimal format:

```
function printToFile(filename_out, dacBits, MSDU)
    fout = fopen(filename_out, 'w'); % use 'w' for text mode
    MSDU = round(MSDU / 2^(16-dacBits)); % come back to 11-bit interval

    % check the correctness of the interval
    MSDU(MSDU < -2^(dacBits-1)) = -2^(dacBits-1);
    MSDU(MSDU > 2^(dacBits-1) - 1) = 2^(dacBits-1) - 1;

```

```
    % Write in unsigned hexadecimal
    MSDU(MSDU < 0) = MSDU(MSDU < 0) + 2^(dacBits);
    fprintf(fout, '%03x\n', MSDU); % write in hex format
    in the file
    fclose(fout);
end
```

After caring about the attenuation, the Matlab code presents some code lines for introducing **noise errors** in the different parts of the frame. As it has been introduced before, the noise errors, differently from the attenuation, have been assumed to be applied also in a single portion of the frame. From a simulation point of view it has been generated a **white gaussian noise** with different magnitudes and applied to the different portion of the frame in order to provide a **set of corrupted frames** of different categories to be analyzed in the simulation environment where then several signals have been observed.

The involved Matlab code lines are the following:

```
%%% PRMBL ERROR %%%
prmbl = I(1:3456); %first 3456 samples are of preamble
fcp = I(3457:end); %the other samples compose the FC and
    payload

noise_prmbl = randn(size(prmbl)).*0.4e4; %medium noise
noise_prmbl_HIGH = randn(size(prmbl)).*2.5e4; %high
   noise
fcp_double = double(fcp);
prmbl_err = noise_prmbl + double(prmbl);
prmbl_err_HIGH = noise_prmbl_HIGH + double(prmbl);

MSDU_prmbl_err = [prmbl_err; fcp_double];
MSDU_prmbl_err_HIGH = [prmbl_err_HIGH; fcp_double];

%%% FC ERROR %%%
noise_fcp = randn(size(fcp)).*0.3e4; %medium noise to
   fcp
prmbl_double = double(prmbl);
fcp_err = noise_fcp + double(fcp);
noise_fcp_HIGH = randn(size(fcp)).*2.5e4; %high noise to
    fcp
fcp_err_HIGH = noise_fcp_HIGH + double(fcp);
```

```
20
21 MSDU_fcp_err = [prmbl_double; fcp_err];
22 MSDU_fcp_err_HIGH = [prmbl_double; fcp_err_HIGH];
23
24 %%% PAYLOAD ERROR %%%
25 payload_seg = I(18484:end);
26 noise_pl = randn(size(payload_seg)).*0.5e4; %medium
      noise error
27 pl_err = noise_pl + double(payload_seg);
28 I_double = double(I(1:18483));
29
30 noise_pl_HIGH = randn(size(payload_seg)).*2.5e4; %high
      noise error
31 pl_err_HIGH = noise_pl_HIGH + double(payload_seg);
32
33 MSDU_payload_err = [I_double; pl_err];
34 MSDU_payload_err_HIGH = [I_double; pl_err_HIGH];
```

As it can be denoted for every portion of the frame the process has been to generate a gaussian noise vector of the same length of the involved portion by using the **randn** function of Maltab that extracts random numbers from the **Normal Distribution**. Then this noise has been added to the original portion of the frame in order to corrupt only the wanted samples. For every case it has been generated two magnitude versions of error, one of the same order of magnitude of the samples of the original frame in figure 4.8 and another of one higher order of magnitude in order to observe any differences. After that code lines, the new calculated arrays with the original frame and its inner portions modified are printed into text files using the same **printToFile** function that has been exposed previously

Regarding the portions of frame that have been considered, as it was already explained it was known a priori that the preamble is an array of 3456 samples from the documentation provided by the company that developed the HPGP subsystem of the Modem IP while since there are no information about the length of the frame control and payload, all the rest of the samples in the MSDU1 frame have been considered to be part of another array named **fcp**. However to have meaningful files for the simulation it was necessary to introduce an error just in the payload portion of the frame. In order to do that, what has been done was just to take a random index in a point of the frame where it is sure to observe the payload (indeed the doubt is only where the payload starts exactly) and so isolate a big segment of it in order to make all the calculations. The exposed Matlab code allowed to create a bunch of text files with modified versions of the original dumped MSDU1 frame

with gaussian noise errors in its different parts. The results are reported in the following figure 4.10.

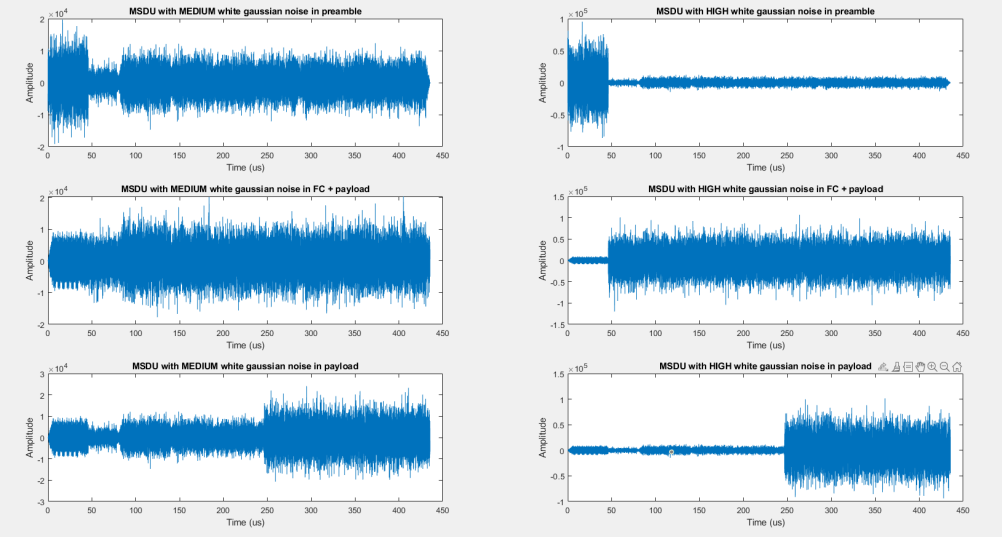

Figure 4.10: Plots of the produced corrupted frames

However, there can be made some observations about the produced corrupted frames for the simulation purposes. First of all one may wonder about why it was used as a basis for the generation of all the wrong frames **just the MSDU1 dumped frame** (the first one on the left in figure 4.6). The reason is that the same procedures have been adopted in order to apply the same modifications also to the other frames and other corrupted frames have been obtained for being simulated later, however since the observations on the signals that will be done later in the thesis are exactly the same, for sake of simplicity just the MSDU1 frame has been taken because it is the one with the shortest payload and so this condition speeds up the result of the simulation that due to its complexity takes several minutes.

Another question that may rise is why **just the white gaussian noise** has been taken as source of error for the frames since previously it has been explained that the possible sources of errors are many. Also in this case the choice was made due to **practicality**. As a matter of fact, in an initial phase of development of the Matlab code has been take into consideration the possibility to apply an **impulsive noise** in the different parts of the frame and it has been practically done as it can denoted by looking at the following code snippet:

```
%%% Gaussian Pulse %%%
a = 0.9*1e4;
b = 20*1e3;
c = 1.5*1e4;
x = t1;
gp = a.*exp(-((x-b).^2)./(c.^2));
MSDU_gauss = transpose(gp) + double(I);

figure;
plot(t1./1e3, MSDU_gauss);
title('MSDU with gaussian noise pulse')
xlabel('Time (us)')
ylabel('Amplitude')
```

In figure 4.11 the correspondent plot has been depicted in order to show the applied impulsive disturb.

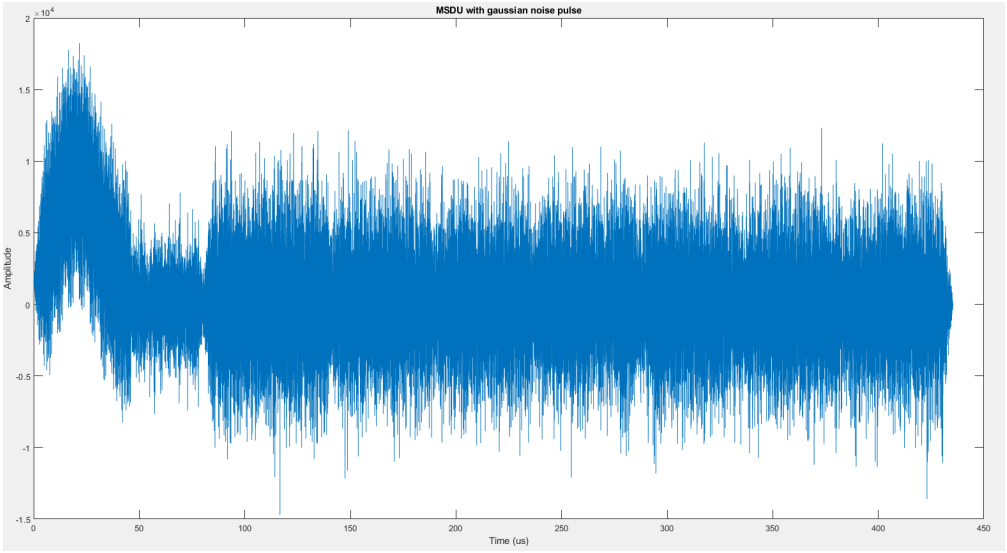

Figure 4.11: Frame corrupted by a gaussian noise pulse in the preamble

In the code above, it has been set the parameters of a Gaussian pulse to be centered in the preamble of the frame with a great magnitude with respect to the average of the other samples. Since then it has been discovered that the effects on the simulation of the HPGP subsystem are the same as the ones obtained with the application of a white gaussian noise of comparable magnitude, then the rest of the code has been developed applying just this kind of disturb to the frames. In

addition, since the HPGP communication is very robust due to some of its features explained in chapter 2 such as the **Data Forward Error Correction** and the **ROBO modes**, in order to corrupt a frame, a very large part of its samples have to be corrupted, this is the reason why in the cases above all the preamble and frame control samples along with a very large segment of the payload have been taken as intervals to be corrupted by noise. If a gaussian noise pulse would have been applied to the different parts of the frame in order to corrupt them, its parameters a,b and c would have been tuned for every case in order to corrupt properly a large part of samples for that specific portion. Since it would be time expensive and useless for the completeness of the test, that can be achieved in the same way by applying just the white gaussian noise instead.

4.4.5 The application of the wrong frames in the simulation environment

After having explained all the process that led to the generation of corrupted frames that are necessary in order to obtain a meaningful and complete simulation of the HPGP subsystem, the next step of the Monitor IP workflow was to apply those frames to the simulation environment in order to make them to be received by the receiving device and so observe the behavior of those signals that can give information about a possible error condition.

As it was introduced before, the block of the simulation environment that is in charge of providing to the receiving IP the wanted frames is the **tb_matrix** one whose most meaningful code lines are reported in the following snippets:

```
architecture rtl of tb_matrix is

type FRAME is (MSDU1, MSDU1_prmbl_err_HIGH, MSDU1_prmbl_err_PURE,
               MSDU1_fcp_err_HIGH, MSDU1_payload_err_HIGH,
               MSDU1_amp_80dB, MSDU1_att_6dB, MSDU1_att_40dB);

signal DATA_IN_int : std_logic_vector(13 downto 0);
signal zc_clk : std_logic;
signal end_file : std_logic;
signal frame_kind : FRAME;
```

Here are the declarations of the signals where the most important one is **frame_kind** of the custom type **FRAME** that allows to visualize directly in the simulation the type of frame that is going to be simulated while in the lines below there is the process that instantiates the clock signal at 75 MHz since this block

has to simulate the operation done by ADC that provides to the receiving IP data samples at that frequency:

```
DATA_IN_V_CLK_p : process
begin
  loop
    DATA_IN_V_CLK <= '0';
    wait for 6.667 ns;
    DATA_IN_V_CLK <= '1';
    wait for 6.667 ns;
  end loop;
end process DATA_IN_V_CLK_p;
```

Then there is the **reading_process** where a file with the frame samples is opened, in this case **MSDU1_payload_err_HIGH.txt** which contains the frame corrupted by an high magnitude white gaussian noise in a payload segment. From this file the samples are stored in the **v_data_read** signal and then assigned to the **DATA_IN** output that has a 14 bits dimension, so as said previously there is the need to make a zero-padding on the LSBs so that the informative content of the frame remains stored in the most significant 11 bits:

```
read_frame_p : process

file file_handler : text;

Variable row         : line;
Variable v_data_read : std_logic_vector(11 downto 0);

begin
  end_file <= '0';
  DATA_IN <= (others => '0');

  -- introduced delay
  for idx in 0 to 99 loop
    wait until (DATA_IN_V_CLK'event and DATA_IN_V_CLK = '1');
  end loop;

  file_open(file_handler, "./tests/hpgp_monitor_rx/
                           MSDU1_payload_err_HIGH.txt",read_mode);
  frame_kind <= MSDU1_payload_err_HIGH;

  while not endfile(file_handler) loop
    readline(file_handler, row);
    hread(row, v_data_read);
    wait until (DATA_IN_V_CLK'event and DATA_IN_V_CLK = '0');
```

```
    DATA_IN_int <= v_data_read & "00";
    DATA_IN <= DATA_IN_int(12 downto 0) & '0';
  end loop;

  end_file <= '1';
  file_close(file_handler);
  DATA_IN <=  (others => '0');
```

Then since there is need to read and simulate also other frames in order to have a meaningful simulation with a significant amount of different cases, a for-loop has been implemented:

```
for i in 0 to 9 loop
  wait for 1063740 ns;

  end_file <= '0';
  DATA_IN <= (others => '0');

  -- introduced delay
  for idx in 0 to 99 loop
    wait until (DATA_IN_V_CLK'event and DATA_IN_V_CLK = '1');
  end loop;

  if i = 0 then
        file_open(file_handler, "./tests/hpgp_monitor_rx/
                            MSDU1_att_40dB.txt",read_mode);
        frame_kind <= MSDU1_att_40dB;
  elsif i = 1 then
        file_open(file_handler, "./tests/hpgp_monitor_rx/
            MSDU1_att_6dB.txt",read_mode);
        frame_kind <= MSDU1_att_6dB;

  - - -

  elsif i = 8 then
        file_open(file_handler, "./tests/hpgp_monitor_rx/
                            MSDU1.txt",read_mode);
        frame_kind <= MSDU1;
  else
        file_open(file_handler, "./tests/hpgp_monitor_rx/
            MSDU1_att_40dB.txt",read_mode);
        frame_kind <= MSDU1_att_40dB;
  end if;
```

```
  while not endfile(file_handler) loop
    readline(file_handler, row);
    hread(row, v_data_read);
    wait until (DATA_IN_V_CLK'event and DATA_IN_V_CLK = '0');
    DATA_IN_int <= v_data_read & "00";
    DATA_IN <= DATA_IN_int(12 downto 0) & '0';
  end loop;

  end_file <= '1';
  file_close(file_handler);
  DATA_IN <=  (others => '0');

end loop;
```

where it has to be denoted that there must be a delay of about 1.06 ms from the reading of a frame and the following one because it has been discovered empirically that that is the correct time that the system takes in order to initialize its parameters between a frame and another one. Moreover in the above code snippet not all iterations have been reported for sake of readability, also because all the frames will be clearly visible when dealing with the results of the simulation.

At this point that all the conditions for having a complete simulation have been set, it has been possible to effectively start the simulation environment in order to make observations on signals that vary according to the different characteristics of the applied frames. In particular, the following figure 4.12 presents the most important results given by the simulation:

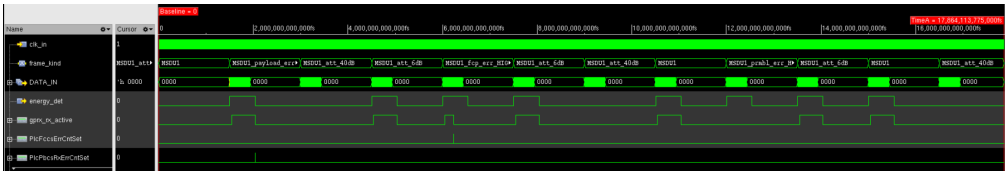

Figure 4.12: Simulation of the HPGP subsystem with the application of the corrupted frames

It is important to make the reader aware that despite it should appear a very simple waveform, the signals that are represented are the result of a very long analysis of every block that compose the HPGP subsystem in order to find some signal that can read the differences between the frames that are received. So at the end of this analysis the most meaningful signals that can be exploited in order to implement monitoring capabilities are those that are visible in the above figure

and will be explained in the following lines.

The **energy_det** is a signal coming out from the **Automatic Gain Controller (AGC)** block of the PHY. This block is mainly responsible of computing the **Received Signal Strength Indicator (RSSI)** which is a parameter that is commonly used when dealing with telecommunications and as its name suggests it measures the strength of the received signal in terms of **power**. The AGC block, as it has been just introduced, is basically the first block in the receiver chain. This because it has to regulate the analog gain of the VGA in the AFE in order to receive the signals in the best dynamic to make the other blocks of the chain elaborate them better. From the specifications reported in chapter 3 the VGA has a gain that is between -18 dB and 48 dB. For doing that it exploits the results given by the computation of the RSSI that if it is below a certain threshold indicated by the constant signal **EN_TH[7:0]** in the code, the signal results to be too much attenuated to be amplified and so the signal does not appear to be received.
The RSSI value is computed through a quite complex algorithm that makes firstly every input sample to be multiplied by a constant that amplifies the magnitude of the sample itself that than is resized by mean of a subtraction of another constant. The result is then passed through an internal block of the AGC that is a **lg10_lut** which basically provides an approximated result of the log10 calculation of its input. The output of the LUT is then multiplied by other constants and rounded before having the definitive RSSI value of the initial input sample. As it can be observed the result of the RSSI calculation is not fully transparent, because it measures the **relative quality** of the signal that has been received and not its absolute value in terms of dBm. As a matter of facts, every manufacturer can decide its own RSSI scale and the correspondence of RSSI values with absolute signal power ones. In this case the company that provided the HPGP subsystem to ST did not provide any kind of explanation regarding the choice of the constants used for computing the RSSI values of the sample as well as the constant values of the **thresholds** that have been used in the code as the already mentioned EN_TH[7:0] signal that has the hexadecimal value of 0x1C (28 in decimal format). What can be deduced also by the few comment that are present in the code is that this way to obtain the RSSI parameter has been obtained by a **Matlab/Simulink** model that simulates the specific characteristics of the device, providing ad-hoc constants for the computation at RTL level. Regarding the value of the threshold EN_TH, it has probably been decided starting from some tests made while the company was developing the system and so based on direct observation.
So what the signal **energy_det** simply indicates is if it has been detected a signal with a certain power. In particular the AGC block computes a **rssi_ma[7:0]** signal which represents the moving average of the punctual rssi[7:0] signal evaluated for every sample calculated in a window of 32 samples. That signal arises and then stays high when that rssi_ma signal gives an energy value that is above the

previously mentioned threshold, on the contrary it stays low as it can be seen by looking at figure 4.13.

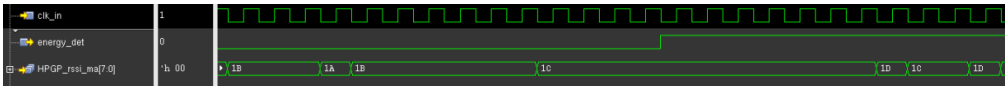

Figure 4.13: Behavior of the energy_det signal when the rssi_ma[7:0] signal is above the EN_TH threshold

As a further proof, by looking at the waveforms in figure 4.12, when the received frame is strongly attenuated (in the test 40 dB), since the power of the signal is very low energy_det does not arise.
The other meaningful signal that has been observed is the **gprx_phy_active** one. As the previous analyzed signal, it comes out from the PHY layer of the Baseband of the HPGP subsystem but specifically it is the output of another block that is the **rx_sync** one. This block has the role to identify the **boundaries of the received frames** and to **synchronize** the **RX PHY chain** by sensing a specific sequence for the **preamble**. As a matter of fact, this synchronization block includes a module that, when receiving a frame, computes the **cross correlation** of the preamble by making an "xor" operation with a stored and well known **preamble reference** that has to be the correct preamble for all the received frames. If the received preamble is the expected one, at the end of the 3456 samples that compose it, the gprx_phy_active signal arises and stays high until the end of the received frame.
As it can be seen from figure 4.12 the signal in analysis stays low when:

1. **energy_det = 0**, because the frame is not received by the system due to the low energy of the signal.

2. a **preamble error** occurs, in this case the system may detect the frame if the energy is sufficient but it is not able to start the demodulation of the symbol because it is not able to synchronize with the received data.

3. a **frame control error** occurs, as a matter of fact despite the preamble has been detected and the system has been synchronized, if there is such an error the gprx_phy_active signal is reset to zero by the **PlcFccsErrCntSet** signal that highlights this kind of error.

The other two signals that have to be discussed to complete this analysis are the just mentioned **PlcFccsErrCntSet** and **PlcPbcsRxErrCntSet**. These two signals, differently form the others that have been just analyzed, come from the MAC layer and specifically from its **PLC_RX** block that is responsible of

receiving the demodulated data from the PHY from which it extracts the correct information to deliver to the rest of the system. In addition, this block checks the integrity of the data that has been received and do that by mean of the **Cyclic Redundancy Check (CRC)** that is a common method that is used in communication networks in order to discover if the transmitted data have been corrupted during the communication. In particular the method involves both the transmitter and the receiver. From the transmitter side, the data packet to be transmitted, so the frame control and the payload, are padded with a number of zeros that is equal to "b-1" where "b" is the number of bits of the binary representation of the specific **generator polynomial**. For the case of the frame control, since it is used a **CRC-24**, that polynomial is:

$$G(x) = x^{24} + x^{23} + x^6 + x^5 + x + 1$$

, as it is stated in the HPGP standard [4], to which it corresponds its binary representation:

$$G = 25'b1100000000000000000001100011$$

where the 1s are present only where the terms of the polynomial have 1 as coefficient. It is analogous for the payload, but this time a **CRC-32** has been applied so the polynomial is:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

to which this time corresponds the binary representation:

$$G = 32'b10000010011000001000111010110111$$

At this point, after the padding of the data packet to be transmitted, it must be divided by the binary representation of the generator polynomial obtaining a **remainder** that than is appended at the end of the data packet. At this point this extended version of the data is again divided by the binary representation of the polynomial generator at the receiver side. If on the receiver side the remainder is 0, the received data results **error-free** and so the signal in analysis, that is one between PlcFccsErrCntSet and PlcPbcsRxErrCntSet depending if the received data is frame control or payload, remains at its reset value so logical zero, otherwise if the remainder is different from 0 it means that some error occurred in the communication network and so the proper signal arises to logical one. The PLC_RX block of the MAC on the receiver side makes use of two modules named **FCCS_CHK** and **PBCS_CHK** that are in charge of computing the division operation as well as check if the remainder is null. It must be said also that, as it has been treated in chapter 2, the payload can be composed by a variable number of **PHY blocks (PBs)** each one has its own CRC, so the PlcPbcsRxErrCntSet signal may rise more times depending on the number of wrong PBs.

As for the other analyzed signals, also the last two are present in the simulation in figure 4.12. As a final consideration on the possible errors in a received frame, it is important to note that there is a sort of **hierarchy of errors**. As a matter of fact the first thing that is considered is the **energy of the received frame**, if it is not sufficient the frame is not seen by the system, so maybe it can be wrong but any error is not highlighted by any signal. Later on, there is the **synchronization** of the receiving chain with the frame and if the system is not able to recognize the correct preamble the gprx_phy_active will never rise, that means that the demodulation procedure will never start. Then the considerations about the correctness of the CRC codes on the frame control and payload are made at MAC layer level, so after that the symbol has been demodulated, in addition it can be seen that in figure 4.12 in correspondence with the **MSDU1_fcp_err_HIGH** where an high noise perturbation has been applied to all the frame with exception of the preamble, the only highlighted error is the frame control one because only the PlcFccsErrCntSet signal arises. This because an error in the frame control portion of the frame may corrupt vital information for the received frame manipulation so that considering further errors in the payload would result useless. That is the reason why the receiving window represented by the gprx_phy_active signal is reset after the recognition of a FC error.

4.5 The HPGP monitor architecture

Once that all the preliminary analyses that were necessary in order to develop consciously the **HPGP Monitor IP** have been done, the architecture of the latter can be finally presented.

Figure 4.14 presents an high level block diagram of the Monitor that allows the reader to observe more details with respect to the ones presented in section 4.3 while figure 4.15 highlights the signals that are exchanged between the **top entity** of the Monitor and the other main blocks involved in the monitoring activity.
The idea behind the implementation of the Monitor IP architecture is that the **HPGP_subsystem** should provide to the Monitor the signals that have been discussed in section 4.4.5 (signals in violet of figure 4.15) that have to be exploited in order to count the **total number of received frames** as well as the **number of errors** of the different typologies. In addition to those signals there are also the **debug signals** (the ones in light blue in figure 4.15) that, as it was introduced in section 4.2.1, are signals that take out from the architecture of the HPGP subsystem several intermediate results that may be inherent to the procedure of demodulation done by the PHY layer or to the frame elaboration held by the MAC layer depending

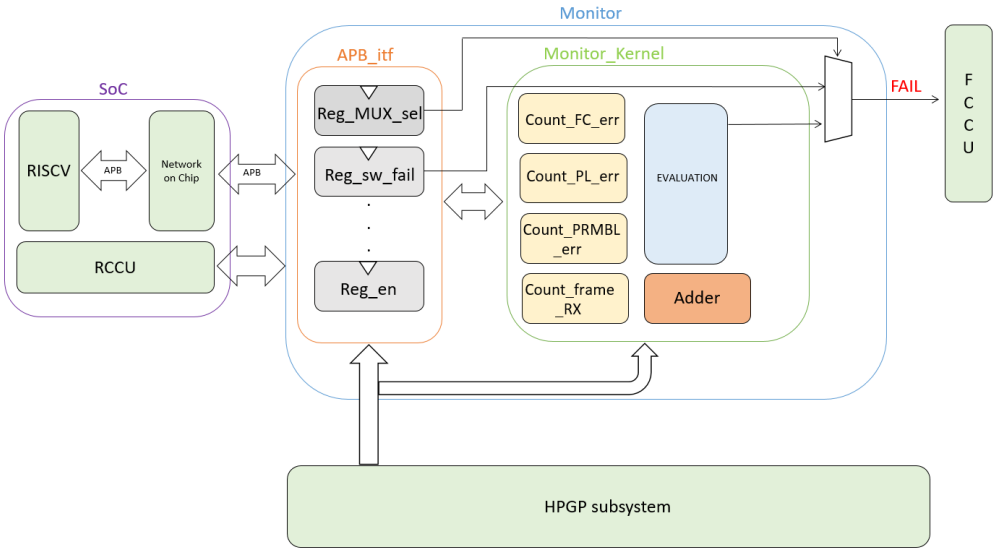

Figure 4.14: High level representation of the HPGP Monitor IP architecture

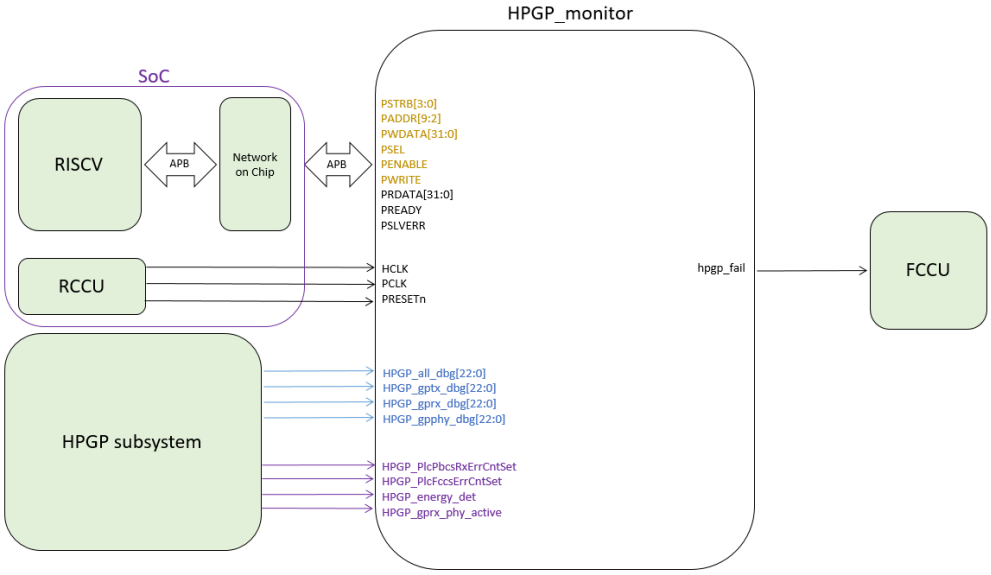

Figure 4.15: HPGP Monitor IP top entity

on how the **muxing logic** is managed at the level of the HPGP subsystem registers that "turn" the multiplexers according to which internal values are needed to be present on these debug signals. In figure 4.16 there is an high level representation of the debug muxing logic that has been developed by the external company. As it can be seen the idea is that there is a wide set of multiplexers whose selection

signals are ruled by the values of some of the HPGP subsystem registers (registers addresses in red are just an example). The **HPGP_all_dbg[22:0]** is the debug signal that is at the end of the debugging chain while the others gather signals at a more inner level. The reason why there are four debugging signals as inputs of the monitor is because there may be the need of observing more internal architecture signals at the same time so in this case it is important to have more than 23 bits available for the purpose. These signals are of strong importance because if something unexpected go wrong in the system, they gather information about the inner operations of the communication and may help for individuating the source of error.

The other signals that are present in the top entity of the HPGP Monitor are those of the **AMBA APB** that are needed for exchanging data to and from the **RISC-V**. In addition there is the signal **HCLK** that is a clock at 150 MHz in input to the Monitor that is used just in the **Monitor Kernel** module in order to have an higher sensitivity and velocity to reveal the rising and falling edges of the HPGP signals since some of these signals are very short in time and a clock of 75 MHz as the PCLK one would has not been sufficiently fast. This concept will be analyzed in details in section 4.18. Both The PCLK and the HCLK are provided to the IP by the **Reset and Clock Control Unit (RCCU)** along with the **PRESETn** signal while the other APB signals come from the **Network on Chip** unit in the **SoC** of the Modem that exchange information directly with the RISC-V. In the end, the Monitor provides a **fail signal** as output to the FCCU that, as it can be seen in figure 4.14 it can be a **software fail** or an **hardware fail**, depending on the selection signal of the multiplexer that is set by an internal register of the APB interface as well as the software fail signal that highlights a fail condition which can be set directly by the RISC-V.

4.5.1 APB Interface

The **APB Interface** is one of the two main modules that compose the architecture of the HPGP Monitor IP along with the **Monitor Kernel** that will be analyzed in the next section 4.18. In figure 4.17 there is a representation of its input/output scheme.
As its name suggest, the role of this module is to represent an **interface** between the RISC-V and the Monitor itself that is needed for **read** and **write** from and to the configuration registers of the Monitor. This interface implements some procedures that allows to **access the configuration registers** of the Monitor IP in order to take values and assign them to output signals of the APB Interface, or to take input signals of the latter and save them into registers. All the state and control signals are written in the registers depending on the value of the provided address contained in **PADDR** signal and the data in the **PWRITE** one. Subsequently

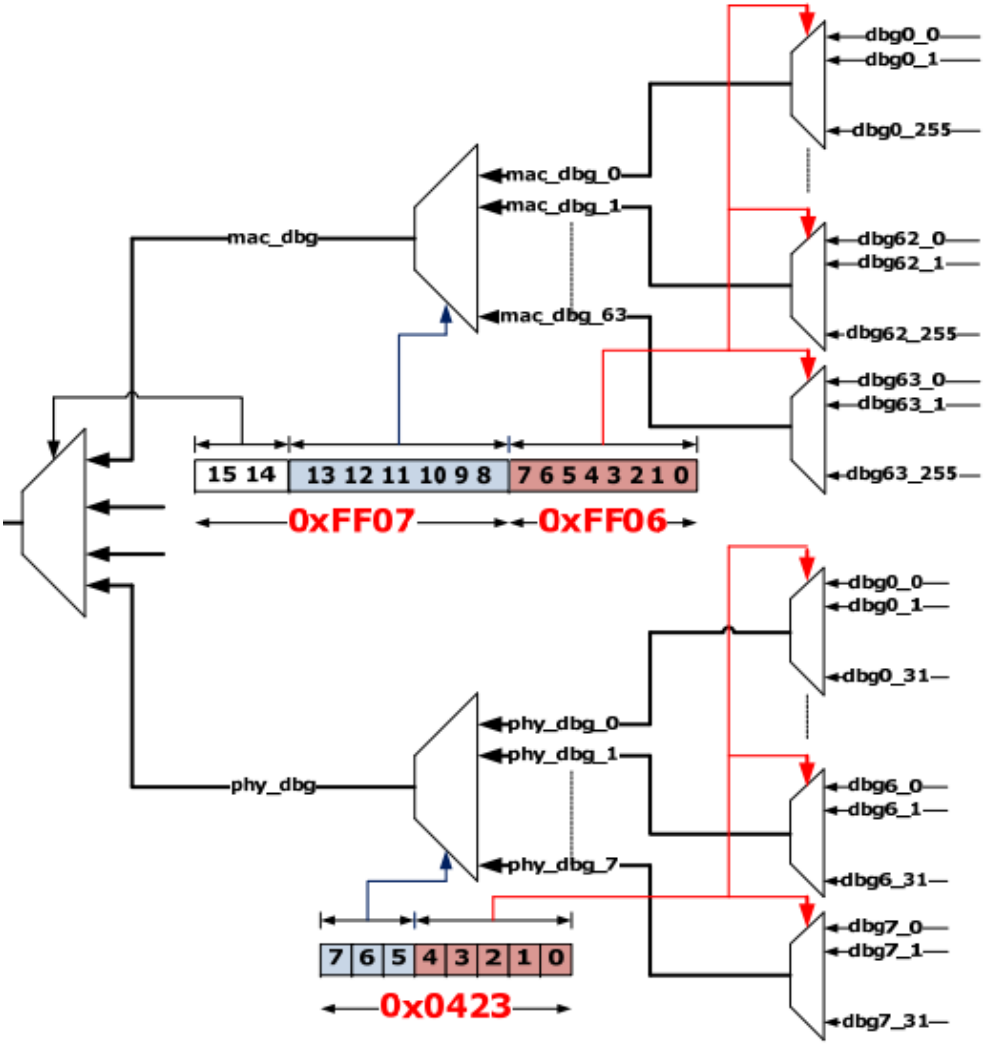

Figure 4.16: High level representation of debug muxing logic (taken from [9])

these registers, whose needed space has been allocated in a proper RAM memory in the system, can be read by mean of the **PRDATA** signal. Looking at figure 4.17, the signals in red represent the **thresholds** that the logic inside the Monitor Kernel have to use in order to recognize if too many errors occurred and so a fail signal must be raised, along with their correspondent **enabling signals**. These signals together with the **monitor_en** signal, **start_frames**, **window**, **MUX_sel** and **SW_fail** are stored in **Read and Write registers** in the memory, as a matter of fact they are wrote in a program in C-language named simply **main.c** that has the

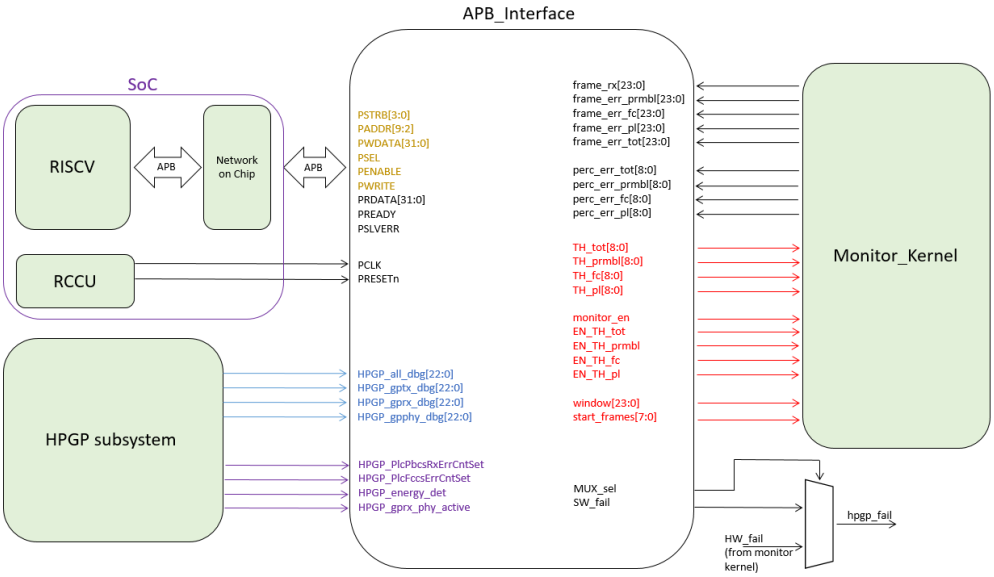

Figure 4.17: APB interface input/output scheme

role to make the whole system to be initiated by mean of the function **initChip()** which initialize all the needed parameters and procedures for making the HPGP subsystem to be ready for the communication.
The signals start_frames and window are instantiated both in the same 32-bit register where the 24 MSBs represent the value of window signal while the first 8 of them represent the start_frame signal. In particular, the latter indicate the number of frames after which the monitor should provide a percentage on the received wrong frames. The meaning behind the introduction of that signal is connected to the fact that if the monitor provides immediately the percentage and there is an error in the first sensed frames, the percentage results to be elevated, with the risk of overcoming the threshold immediately, for this reason it has been thought that computing the percentage after a set number of frames could be a better choice in terms of functionality. The **window signal** instead, represents the number of frames after which the calculation of the percentage restarts and it was introduced mainly because the team was not aware about the precise number of received frames during a real charging operation at this point of the project, so making computations on an **editable** window gave more flexibility. In addition, since a very low number of errors is expected, this would lead to an incredibly low percentage of errors to be detected in the case that it is calculated on the totality of received frames, while choosing a smaller window allows to relax the specification on the parallelism of the **perc_err** signals.
The following VHDL code lines of the APB Interface regard the instantiation of the just mentioned R/W registers:

```
  RW_cfg_reg(monitor_en, reg_N =>  0, bit_N =>  0, reset_to => '0');


  RW_cfg_reg(start_frames, reg_N =>  1, left_bit =>  7, right_bit =>  0,
                                              reset_to => "00000000");
  RW_cfg_reg(window, reg_N =>  1, left_bit =>  31, right_bit =>  8,
                                     reset_to => "00000000000000000000000");

  RW_cfg_reg(TH_tot, reg_N =>  2, left_bit =>  8, right_bit =>  0,
                                              reset_to => "000000000");
  RW_cfg_reg(TH_prmbl, reg_N =>  3, left_bit =>  8, right_bit =>  0,
                                              reset_to => "000000000");
  RW_cfg_reg(TH_fc, reg_N =>  4, left_bit =>  8, right_bit =>  0,
                                              reset_to => "000000000");
  RW_cfg_reg(TH_pl, reg_N =>  5, left_bit =>  8, right_bit =>  0,
                                              reset_to => "000000000");


  RW_cfg_reg(EN_TH_tot ,reg_N =>  6, bit_N => 0, reset_to => '0');
  RW_cfg_reg(EN_TH_prmbl, reg_N =>  6, bit_N => 1, reset_to => '0');
  RW_cfg_reg(EN_TH_fc, reg_N =>  6, bit_N => 2, reset_to => '0');
  RW_cfg_reg(EN_TH_pl, reg_N =>  6, bit_N => 3, reset_to => '0');

  RW_cfg_reg(MUX_sel, reg_N =>  7, bit_N => 0, reset_to => '0');
  RW_cfg_reg(SW_fail, reg_N =>  7, bit_N => 1, reset_to => '0');
```

where the **RW_cfg_reg** function is defined differently if the output is a std_logic or a std_logic_vector:

```
-- procedure to read a R/W system register and assign its value to a
-- std_logic_vector output signal
procedure RW_cfg_reg(signal out_signal : out std_logic_vector;
        constant reg_N, left_bit, right_bit : integer) is begin

        wr_mask_val(reg_N)(left_bit downto right_bit) <= (others => '1');

        v_cfg_rdata(reg_N)(left_bit downto right_bit) :=
        cfg_data_ff(reg_N)(left_bit downto right_bit)
        xor reset_val(reg_N)(left_bit downto right_bit);

        out_signal <= v_cfg_rdata(reg_N)(left_bit downto right_bit);

        check(reg_N, left_bit, right_bit);
```

```
end procedure RW_cfg_reg;

-- same as before but reset to any value
procedure RW_cfg_reg(signal out_signal : out std_logic_vector;
        constant reg_N, left_bit, right_bit : integer;
        constant reset_to : std_logic_vector) is begin

        RW_cfg_reg(out_signal, reg_N, left_bit, right_bit);

        reset_val(reg_N)(left_bit downto right_bit) <= reset_to;

end procedure RW_cfg_reg;

-- procedure to read a R/W system register and assign its value to a
-- std_logic output signal
procedure RW_cfg_reg(signal out_signal : out std_logic;
        constant reg_N, bit_N : integer) is begin

        wr_mask_val(reg_N)(bit_N) <= '1';

        v_cfg_rdata(reg_N)(bit_N) := cfg_data_ff(reg_N)(bit_N)
        xor reset_val(reg_N)(bit_N);

        out_signal <= v_cfg_rdata(reg_N)(bit_N);

        check(reg_N, bit_N);

end procedure RW_cfg_reg;

-- same as before but reset to any value
procedure RW_cfg_reg(signal out_signal : out std_logic;
        constant reg_N, bit_N : integer;
        constant reset_to : std_logic) is begin

        RW_cfg_reg(out_signal, reg_N, bit_N);

        reset_val(reg_N)(bit_N) <= reset_to;

end procedure RW_cfg_reg;
```

As it can be seen, the procedure in analysis reads a subset of bits from a **configuration register**, resets its value if necessary, and then assign the results to the output signal. At the end of the procedure there is also a call to a further **check**

procedure that verifies if the register exists and the specified bits have been defined. On the other hand, the following code lines represent a code snippet of the already mentioned **main.c** file:

```
#include "plc.h"
#define MONITOR_APB_BASE  0x23002000
\\ ...
int main()
{
    \\ ...
    initChip();

    \\ ...

    HPGPMONITOR[0] = 1;          //monitor_en = 1
    HPGPMONITOR[1] = 1025;       //window = 00000000000000000000100 =
     4  start_frames = 00000001 = 1 --> 2^0 + 2^10 = 1025

    HPGPMONITOR[2] = 160;        //TH_tot = 010100000 --> 0.625
    HPGPMONITOR[3] = 96;         //TH_prmbl = 001100000 --> 0.375
    HPGPMONITOR[4] = 0;          //TH_fc = 000000000
    HPGPMONITOR[5] = 0;          //TH_pl = 000000000

    HPGPMONITOR[6] = 3;          //EN_TH_tot = 1 and EN_TH_prmbl = 1;

    HPGPMONITOR[7] = 0;          //MUX_sel = 0 --> HW fail sensed,
    SW_fail = 0

    \\ ...
    return 0;
}
```

So in order to make a summary of all the workflow that involves the APB Interface, in the main.c file the **R/W registers** are written with some values, observable by looking at the comment in the code, that have been chosen for the simulation. Then this code is compiled by the GCC and transformed in **machine code** for being interpreted by the RISC-V that than writes this values in the RAM memory starting from the chosen **base address** 0x23002000 to which an offset multiple of 4 bytes is added. As an example, writing HPGPMONITOR[6] = 3 means to write the value 3 to the 32-bit register located at the address 0x23002000 + 0x18 where 18 in hexadecimal is equal to 6*4 = 24 in decimal format. Later on at RTL level, the RISC-V communicates which registers have to be read through the AMBA APB signals, and then the value of that registers is assigned to the output signals of the APB Interface by mean of the procedure **RW_cfg_reg** discussed before. In this way the APB Interface from the instructions of the processor through the AMBA

bus is capable of taking out the values stored in the registers and assign them to output signals that will be used in the rest of the RTL Monitor architecture.
If one pays attention to the comments that has been attached to the C-code above, it can be noticed that the threshold values that have been specified for each kind of error, are expressed in a **fixed point notation** where the MSB represents the **integer value** while the other 8 bits refer to the **fractional part**. As it will be seen while analysing the **Monitor Kernel** which contains an arithmetic unit that practically computes the percentages for the errors, the latter are expressed in this format because it allows to calculate small percentage values making use of a relatively limited parallelism for the involved signals. This aspect will be recalled while treating the dedicated section.
Regarding the other signals involved in the APB Interface, so those in input from the HPGP subsystem and those in black coming from the Monitor Kernel looking always at figure 4.17, the procedure is the same as the one that has been just described, however that signals are stored in **Read-only registers** because their values are not carried by the RISC-V instructions but other modules of the architecture so the goal is just to read their values from the register. As an example, considering the **frame_rx[23:0]** signal that enters the APB Interface coming from the Monitor Kernel, it just represents the number of received frames in a certain window and it may be wanted to be stored in a dedicated register in order to have its value being accessible via software by mean of the PRDATA[31:0] of the AMBA APB. So in the VHDL description of the APB Interface, all these signals have been involved in the usage of the **RO_cfg_reg** procedure defined as follows:

```
RO_cfg_reg(frame_rx,                reg_N => 20, left_bit => 23,
                                                 right_bit =>  0);

RO_cfg_reg(frame_err_tot,           reg_N => 21, left_bit => 23,
                                                 right_bit =>  0);

RO_cfg_reg(frame_err_prmbl,         reg_N => 22, left_bit => 23,
                                                 right_bit =>  0);

RO_cfg_reg(frame_err_fc,            reg_N => 23, left_bit => 23,
                                                 right_bit =>  0);

RO_cfg_reg(frame_err_pl,            reg_N => 24, left_bit => 23,
                                                 right_bit =>  0);

RO_cfg_reg(perc_err_tot,            reg_N => 30, left_bit => 8,
                                                 right_bit =>  0);
```

```
  RO_cfg_reg(perc_err_prmbl,          reg_N => 31, left_bit => 8,
                                                   right_bit =>  0);

  RO_cfg_reg(perc_err_fc,             reg_N => 32, left_bit => 8,
                                                   right_bit =>  0);

  RO_cfg_reg(perc_err_pl,             reg_N => 33, left_bit => 8,
                                                   right_bit =>  0);


  RO_cfg_reg(HPGP_energy_det,         reg_N => 50, bit_N =>  0);
  RO_cfg_reg(HPGP_gprx_phy_active,    reg_N => 50, bit_N =>  1);
  RO_cfg_reg(HPGP_PlcFccsErrCntSet,   reg_N => 50, bit_N =>  2);
  RO_cfg_reg(HPGP_PlcPbcsRxErrCntSet, reg_N => 50, bit_N =>  3);


  RO_cfg_reg(HPGP_gptx_dbg,           reg_N => 60, left_bit => 22,
                                                   right_bit =>  0);

  RO_cfg_reg(HPGP_gprx_dbg,           reg_N => 61, left_bit => 22,
                                                   right_bit =>  0);

  RO_cfg_reg(HPGP_gpphy_dbg,          reg_N => 62, left_bit => 22,
                                                   right_bit =>  0);

  RO_cfg_reg(HPGP_all_dbg,            reg_N => 63, left_bit => 22,
                                                   right_bit =>  0);
```

where the relative definition of the used procedure is as follows:

```
-- procedure to read a Read only system register from a
-- std_logic_vector input signal
procedure RO_cfg_reg(signal in_signal : in std_logic_vector;
        constant reg_N, left_bit, right_bit : integer) is begin

        v_cfg_rdata(reg_N)(left_bit downto right_bit) := in_signal;
        check(reg_N, left_bit, right_bit);
end procedure RO_cfg_reg;
```

In this case, differently from the previous procedure, the signal is in input to the function, this means that the input signal that comes from another module is assigned to that v_cfg_rdata wire that than is assigned at its turn to the

PRDATA[31:0] signal for being accessed via software as can be denoted by the last code lines:

```
-- assign output read data at the end
 PRDATA_i <= v_cfg_rdata(int_add);
```

while in RW_cfg_reg procedure, the signal was an output of the function, to which v_cfg_rdata was assigned.

4.5.2 Monitor Kernel

In this section will be analyzed the other main module that composes the overall architecture of the Monitor IP so the **Monitor Kernel**. It represents the **core of the Monitor** because with the data that receives both from the HPGP subsystem and the APB Interface it computes several important results, such as the **total number of received frames** and the **number of wrong received frames** making distinctions between the different typology of errors. Furthermore it computes the **percentages of wrong frames**, one per each type of frame corruption, and it is able to recognize if one or more of this percentages overcome the threshold values provided by the APB Interface and set via software using the RISC-V and the AMBA APB. The top entity of the Monitor Kernel is presented in figure 4.18.

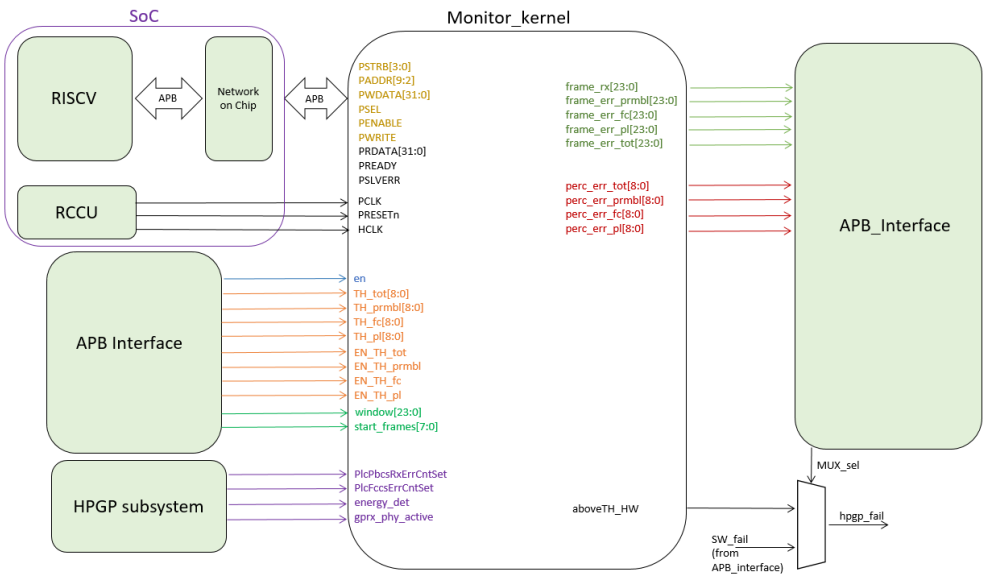

Figure 4.18: Monitor Kernel input/output scheme

From that figure 4.18, a key role is played by the signals in violet that are the signals that have been analyzed in section 4.4.5. This because they are those signals that are analyzed and manipulated by the logic inside the Monitor Kernel in order to extract information about the received frames and eventual error that occurred during the communication. The signals in light orange coming from the APB Interface are the value of the **percentage thresholds** that have been set by software, writing the registers in the **main.c** file mentioned before and they come along with the correspondent **enabling threshold signals**. In practice the Monitor Kernel has been developed in order to sense the received frames and calculate in real time the percentages for each type of error in them.
In truth the computed percentage values are not properly percentages, since they are calculated just by making a division between the errors of a specific kind and the number of received frames for that window while the multiplication by the factor 100 can be easily done via software in order not to increase the instantiated area of the FPGA in phase of synthesis. As an example it can be reported the one that has been considered for the simulation in which are considered basically the thresholds **TH_tot[8:0]**, that is the one regarding the overall number of errors and **TH_prmbl[8:0]** which imposes a limit to the number of preamble errors. In particular, as it can be seen by the previous reported code of the main.c file, the first threshold has been set to a decimal value that is correspondent to the binary fixed point value of 0.625 while the threshold on the preamble errors is of 0.375. In addition the only enabling signals equal to 1 are **EN_TH_tot** and **EN_TH_prmbl**, in this way, only the overcoming of at least one of these 2 specified threshold is taken into consideration and reflects in a **fail condition** while the others are not considered. Resuming, if any of the enabled thresholds is exceeded, the **aboveTH_HW** signal arises, meaning that a failure occurred at **hardware level**.

Regarding the other inputs of the Monitor Kernel, there are the **window[23:0]** and the **start_frames[7:0]** signals that are highlighted in green and as it has been discussed previously represent the "window" of the received frames that is taken into consideration for the percentage calculations and the number of those received frames that has to be awaited before providing a meaningful percentage value at the output. An important mention must be done for the two clock signals **PCLK** and **HCLK** where the first one is at 75 MHz and the latter at 150 MHz. So why there are two different clock domains inside Monitor Kernel? This because, as it will be showed soon, the Monitor Kernel is composed by a set of **counters** that basically count the number of **rising** or **falling edges** of the signals in violet that comes from the HPGP subsystem in order to count the number of received frames and the errors. These signals are mostly **very short pulses** that cover at maximum 3 or 4 clock periods in the PCLK domain, so in order to have a more

reliable detection of the edges the HCLK domain has been introduced just for the counters that must perform this operation of **edge detection**. This does not represent a problem in the architecture because the data flows from an higher clock domain to a lower one, since the last inner module in the data workflow is a block named **EVALUATION** that just performs the calculation of the error percentages and rises the aboveTH_HW signal if necessary.

The absolute value of the received frames and that one of the different typology of wrong frame along with the error percentages are the outputs of the Monitor Kernel, as it can be denoted by looking at figure 4.18 the red and green signals. These signals are in input to the APB Interface and stored in **read only registers** as explained in the previous section.
Regarding the **hardware fail signal** named aboveTH_HW, it is also an output of the Monitor Kernel but it does not enter the APB Interface but a **Multiplexer** that takes as inputs that signal and the **software fail signal** set in the APB Interface as well as the selection signal of the multiplexer.

In the following figure 4.19 there is a representation of the inner architecture of the Monitor Kernel:

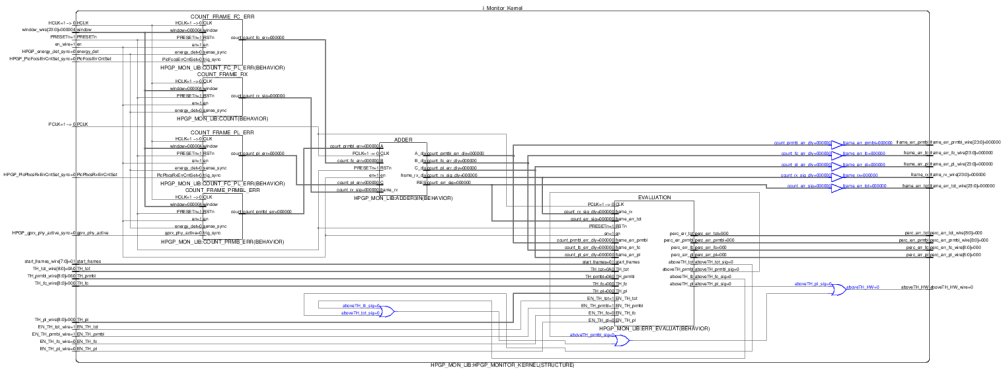

Figure 4.19: Inner architecture of the Monitor Kernel

From that figure there can be observed the main blocks composing the architecture as well as the connections between them. Given the amount of details in figure 4.19, in order to analyze better the blocks there will be showed zoomed versions of the architecture. Note that this figures with the block scheme of the architecture have been taken directly from **Cadence SimVision**, the tool that has been used in order to simulate the IPs, so the numeric values that can be seen as assigned to the input/output signals of the inner modules have a meaning in a particular simulation of the Monitor IP that will be analyzed in the next section 4.5.3. Now the attention has been dedicated just to the blocks architecture and the connections between them.

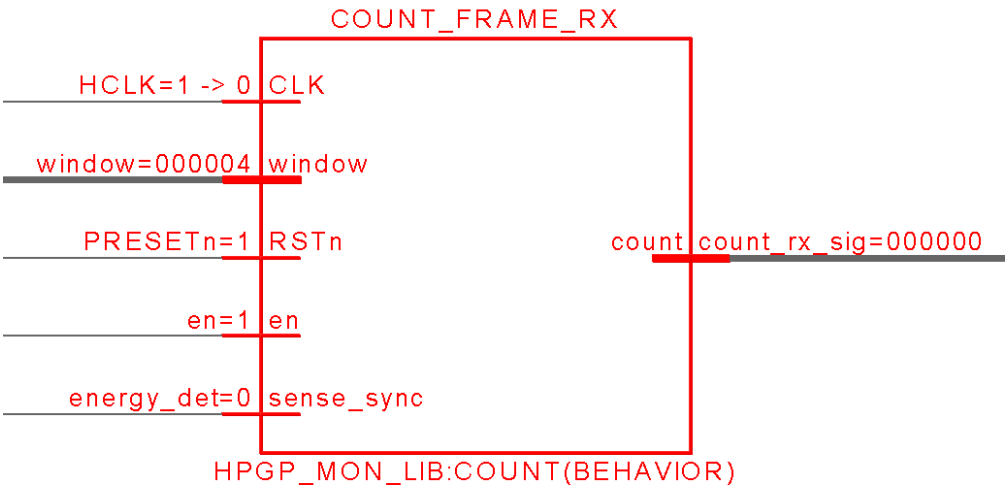

Figure 4.20: View on the counter of the received frames

Figures 4.20 and 4.21 highlight the counters that are needed to count respectively the **received frames** and the **wrong received frames** making a distinction based on which portion of the received frame has been corrupted. Going into the implementation details, the four instantiated counters have been realized making use of three instances of counter that have some similarities but that also differ between them. Their differences are due to the different kind of error that they have to count. As it was introduced in section 4.4.5, a frame is received when the **energy_det** signal is high, so what the **COUNT_FRAME_RX** counter (that is an instance of the **COUNT** entity) must do is just to sense the presence of the pulse of that signal and then increase its internal count. That counter is simply composed by two processes where the first one, **sense_FF**, consists in the behavioral implementation of a Flip Flop that samples the input **sense_sync** signal which is connected to the external **energy_det** signal in such a way that its old value (of 1 clock cycle) can be stored for being used in the detection of a **falling edge** along with the actual sense_sync value. Then the **pulse_fall** signal is used in the other process **counting** for incrementing the value of the counter that senses the received frames. In the following code snippet are reported the most meaningful code lines of the COUNT component:

```
sense_FF: process(CLK, RSTn)
begin
    if RSTn = '0' then -- Reset asincrono
```

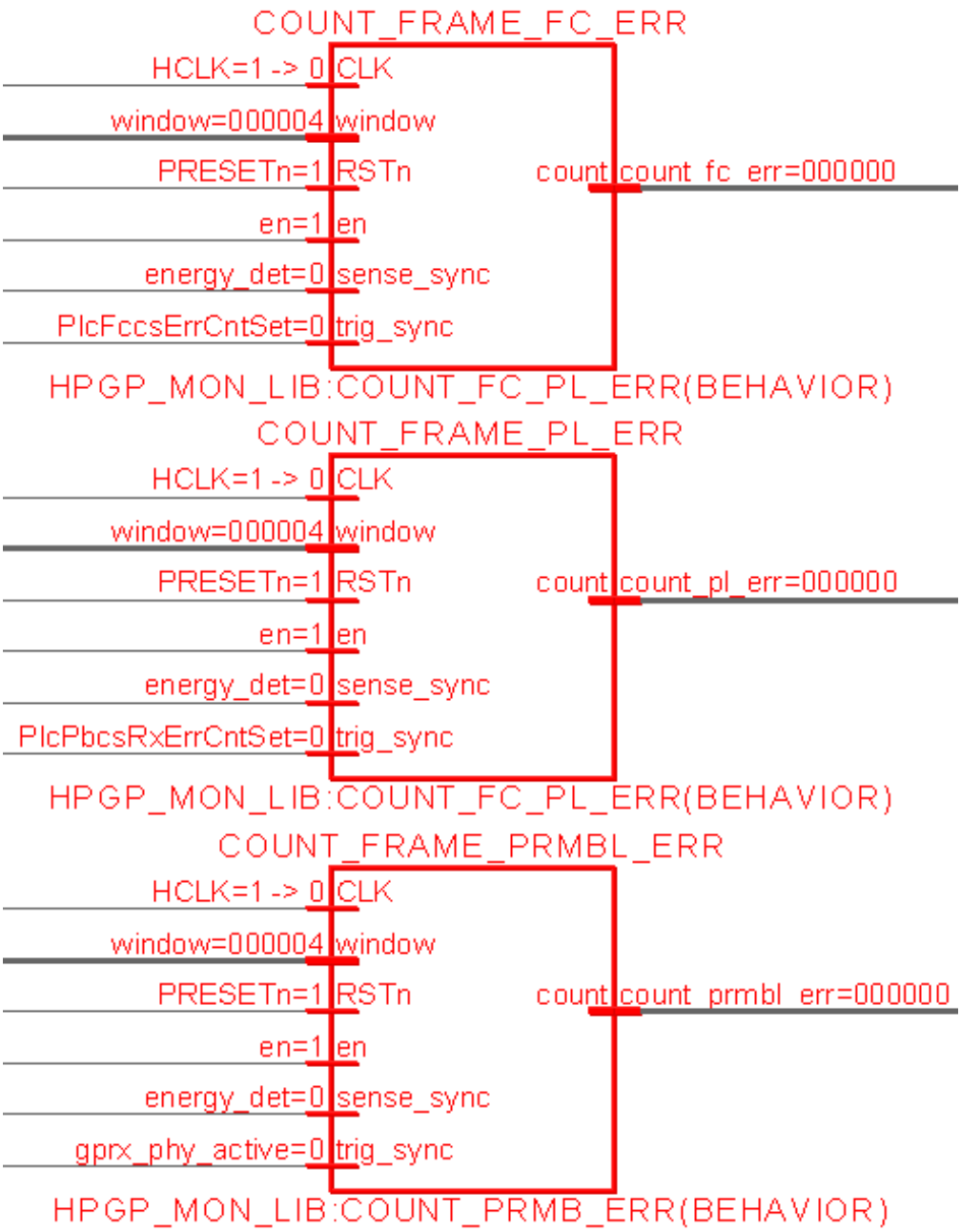

Figure 4.21: View on the error counters in the Monitor Kernel architecture

```
    sense_sync_edge <= '0';
elsif rising_edge(CLK) then
    sense_sync_edge <= sense_sync;
end if;
```

```
end process sense_FF;


 pulse_fall <= sense_sync_edge and (not sense_sync);


counting: process(CLK, RSTn)
begin
    if RSTn = '0' then -- Reset asincrono
        CNT_val <= (others => '0');

    elsif rising_edge(CLK) then

        if en = '1' then
            if pulse_fall = '1' then

                if unsigned(CNT_val) = unsigned(window) then
                    CNT_val <= "000000000000000000000001";
                else
                    CNT_val <= CNT_val + 1;
                end if;

            end if;
        end if;
    end if;
end process counting;

count <= std_logic_vector(CNT_val);
```

Regarding the implementation of the other counters, the idea is basically the same although there are differences in how errors manifest themselves through the signals from HPGP subsystem. As a matter of fact, if a frame is corrupted by a **preamble error**, the energy_det signal must be high since the frame, despite it is wrong, it has been received. However in that time window set by the energy signal, the **gprx_phy_active** signal stays low since the synchronization between the received frame and the demodulation modules could not be possible due to the preamble error. Given that information, the counter must increment its value if it senses the energy_det signal and does not sense the gprx_phy_active one while the first stays high:

```
sense_trig_FF: process(CLK, RSTn)
begin
    if RSTn = '0' then -- Reset asincrono
        sense_sync_edge <= '0';
```

```
        trig_sync_edge <= '0';
    elsif rising_edge(CLK) then
        sense_sync_edge <= sense_sync;
        trig_sync_edge <= trig_sync;
    end if;
end process sense_trig_FF;


pulse_sense_fall <= sense_sync_edge and (not sense_sync);
pulse_trig_rise <= not trig_sync_edge and trig_sync;


counting: process(CLK, RSTn)
begin
    if RSTn = '0' then -- Reset asincrono
        CNT_val_err <= (others => '0');
        CNT_val_frame <= (others => '0');
        flag_trig <= '0';

    elsif rising_edge(CLK) then

        if en = '1' then

            if sense_sync = '1' and pulse_trig_rise = '1' then
                flag_trig <= '1';
            end if;


            if pulse_sense_fall = '1' then

                -- count internally the received frames --> if the window
                --th. is overcome then restart the counting from 0
                if unsigned(CNT_val_frame) = unsigned(window) then
                    CNT_val_frame <= "00000000000000000000000001";

                    if flag_trig = '1' then
                        flag_trig <= '0';
                        CNT_val_err <= (others => '0');

                    elsif flag_trig = '0' then
                        flag_trig <= '0';
                        CNT_val_err <= "00000000000000000000000001";
                    end if;
```

```
                -- depending from the trigger pulse increase or not
                --the count of the errors
                else
                    CNT_val_frame <= CNT_val_frame + 1;

                    if flag_trig = '1' then
                        flag_trig <= '0';

                    elsif flag_trig = '0' then
                        flag_trig <= '0';
                        CNT_val_err <= CNT_val_err + 1;
                    end if;

                end if;
            end if;
        end if;
    end if;
end process counting;


count <= std_logic_vector(CNT_val_err);
```

As it can be seen from the code snippet above of the **COUNT_FRAME_PRMB_ERR**, the same method for detecting the edges of signals has been applied but this time not only is detected the falling edge of the **energy_det** signal but also the rising edge of the **trig_sync** signal which in this case is connected to the **gprx_phy_active** signal for detecting a possible preamble error. So if the falling edge of the energy signal is detected and no rising edges of the active one have been detected in the meanwhile, then there is a preamble error and the count increments. It has to be denoted that in this case it is necessary to make the counter increase its value in correspondence of the falling edge of the energy signal, because it must awaited for being sure that no active signal arose during its time window. In the previous case of the counter for the received frames, since just energy_det must be sensed, the count could be incremented on its rising edge. However in order to have all the results of the counters being synchronized, as a time reference it was used the falling edge of the energy signal for all the cases.

As a matter of fact also for the counters **COUNT_FRAME_FC_ERR** and **COUNT_FRAME_PL_ERR** which use the same implementation, the procedure is the same and the counter is incremented (if it should be) after the detection of the falling edge of the sense input signal that is always connected to the energy_det one. However this time what changes is the trig signal that, in the case of the **frame control** error detection, is connected to **PlcFccsErrCntSet** while

in the **payload** error counter is connected to **PlcPbcsRxErrCntSet** as it can be seen by looking at figure 4.21. Also the implementation is slightly different because this time the increment of the counter should happen if a rising edge of the trig signal is detected during the energy time window. For sake of completeness, the code lines that changes with respect to the previous case are reported:

```
-- count internally the received frames --> if the window th. is overcome
--then restart the counting from 0
if unsigned(CNT_val_frame) = unsigned(window) then
    CNT_val_frame <= "00000000000000000000000001";

    if flag_trig = '0' then
        flag_trig <= '0';
        CNT_val_err <= (others => '0');

    elsif flag_trig = '1' then
        flag_trig <= '0';
        CNT_val_err <= "00000000000000000000000001";
    end if;


-- depending from the trigger pulse increase or not the count of the
--errors
else
    CNT_val_frame <= CNT_val_frame + 1;

    if flag_trig = '0' then
        flag_trig <= '0';

    elsif flag_trig = '1' then
        flag_trig <= '0';
        CNT_val_err <= CNT_val_err + 1;
    end if;

end if;
```

As a final consideration on the counters, one may notice that there is any signal which manages eventual **overflow conditions**. Despite it may be considered an error at a first view, in practice managing the overflow of the counters of this IP is useless because they can count a number of frames that is limited to the value of the**window** that has been defined, so every time that a number of received frame equals the extension of the window all the counters reset their values, so an additional control on the overflow is definitely not needed. This is also one of the reasons why the concept of window has been introduced in the design, because the number of the total received frames in a real application cannot be known and

for this motivation it would be difficult to manage properly an overflow condition. It also has to specified that the number of bits reserved to the window signal are 24 but in practice windows of about 1024 frames will be used in the real FPGA implementation, however such an high number of bits allow a better flexibility.

Going on with the analysis of the Monitor Kernel, the values of the counters become the inputs of an **adder** that has the role to sum the number of errors between them in order to get a **total number of received wrong frames** as can be seen in figure 4.22. An important note on the adder is its inner signal **add_dly** that has been inserted for introducing a degree of freedom in terms of **clock cycles** that the unit takes for providing its outputs that not only consist in the sum but also in the values of the counters in input that are delayed for being synchronized with the sum itself. This feature has been added for introducing **multicycle** constraints during the synthesis on FPGA in order to relax the timing specification for avoiding a **negative slack**. As it can be seen from the code below it is a 1-bit signal so actually it can provide at maximum 1 extra clock but it was sufficient for the synthesis.

```
RES_sig <= unsigned(A) + unsigned(B) + unsigned(C);

add: process(CLK, RSTn)
begin
    if RSTn = '0' then -- Reset asincrono
        RES_reg                 <= (others => '0');
        frame_rx_dly         <= (others => '0');
        A_dly                        <= (others => '0');
        B_dly                        <= (others => '0');
        C_dly                        <= (others => '0');
        add_dly                 <= '1';
    elsif rising_edge(CLK) then
        if en = '1' then
            if add_dly = '0' then
                RES_reg <= RES_sig;
                frame_rx_dly <= frame_rx;
                A_dly <= A;
                B_dly <= B;
                C_dly <= C;
                add_dly <= '1';
            else
                add_dly <= '0'; --add_dly - 1;
            end if;
        end if;
    end if;
end process add;
```

```
RES <= std_logic_vector(RES_reg(23 downto 0));
```

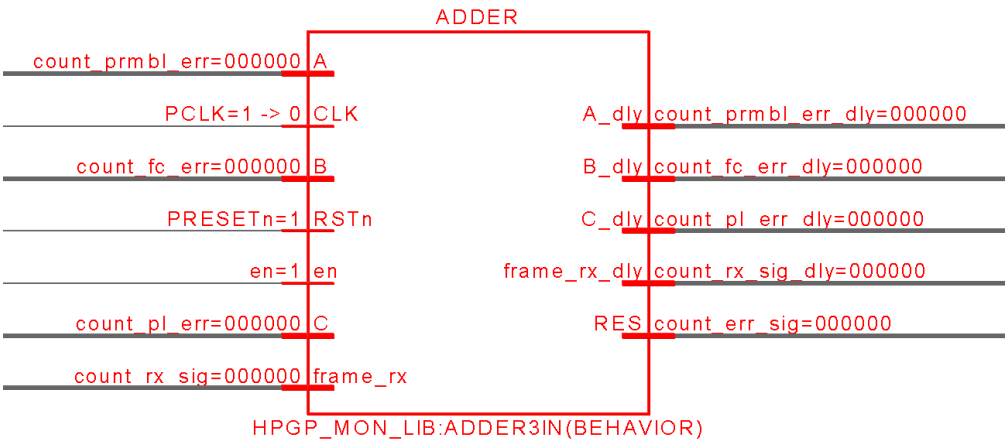

Figure 4.22: View on the adder unit

Continuing with the description of the Monitor architecture, in figure 4.23 is presented the view on the just mentioned **EVALUATION** module that is in charge of performing the following operations:

- It **computes the percentage values** of the total number of errors on the total received frames in the specified window as well as the same for every kind of error, exploiting the inputs provided by the counters.
- It **recognizes the overcoming of the thresholds** provided directly by the APB Interface along with their enabling signals that communicate to the unit to which threshold being sensible. There is a specific **"above"** signal for every kind of error.

Here are reported the code lines of the module that implement the calculations for the percentage values of the errors over the number of received frames. Note that only the code lines regarding the calculation of the percentage of the total errors have been reported since those regarding the other error typologies are the same:

```
process(frame_rx, start_frames, frame_err_tot, frame_err_prmbl, frame_err_fc,
        frame_err_pl, en)
begin
    if en = '1' then
        if unsigned(frame_rx) > unsigned(start_frames) then
```

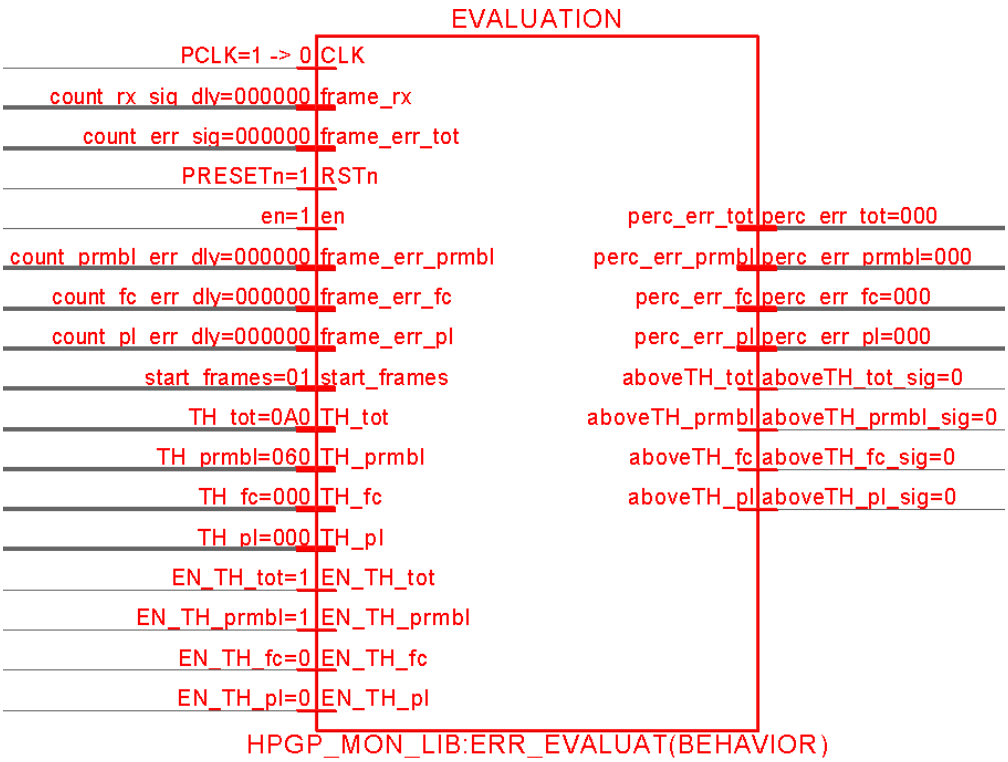

Figure 4.23: View on the "EVALUATION" module

```
            perc_err_tot_sig    <= to_unsigned(to_integer(unsigned(frame_err_tot
                                   & "00000000"))
                                   / to_integer(unsigned(frame_rx)), 9);
            . . .
        else
            perc_err_tot_sig    <= (others => '0');
            . . .
        end if;
    else
        perc_err_tot_sig    <= (others => '0');
        . . .
    end if;
end process;

process(CLK, RSTn)
begin
    if RSTn = '0' then
        perc_err_tot_reg <= (others => '0');
        . . .
```

```
        aboveTH_tot <= '0';
        . . .

        div_dly <= "01";

    elsif rising_edge(CLK) then
        if en = '1' then
            -- just the division to calculate
            if unsigned(frame_rx) > unsigned(start_frames) then
                if div_dly = "00" then
                    -- number of the errors are multiplied by 256 to represent
                    -- correctly the fractional part
                    perc_err_tot_reg <= perc_err_tot_sig;
                    . . .
                    div_dly <= "01";
                else
                    div_dly <= div_dly - 1;
                end if;
            else
                perc_err_tot_reg <= (others => '0');
                . . .

                aboveTH_tot <= '0';
                . . .

                div_dly <= "01";
            end if;

            -- thresholds evaluation
            if EN_TH_tot = '1' then
                if perc_err_tot_reg > unsigned(TH_tot) then
                    aboveTH_tot <= '1';
                else
                    aboveTH_tot <= '0';
                end if;
            end if;

            if EN_TH_prmbl = '1' then
                if perc_err_prmbl_reg > unsigned(TH_prmbl) then
                    aboveTH_prmbl <= '1';
                else
                    aboveTH_prmbl <= '0';
                end if;
            end if;
```

```
            . . .

        else
            perc_err_tot_reg <= (others => '0');
            . . .

            aboveTH_tot <= '0';
            . . .
        end if;
    end if;
end process;

perc_err_tot  <= std_logic_vector(perc_err_tot_reg);
. . .
```

A note must be done for the way the **division** is performed. As it has been introduced in the previous lines, the percentage that comes out from the division is in **fixed point** notation where the MSB represents the integer part while the remaining 8 bits the fractional part. In this way it is possible to represent an error percentage from 0 to 1 (of course in SW the value should be multiplied by 100) in steps of 0.0039. This methodology for managing the division operation has been chosen because it represented the best option for an **implementative** point of view since it allows to have a quite granular representation of the division result without adding too much complexity in terms of allocated area.
Another thing that has to be noticed is the presence of the **div_dly** signal that, as in the case of the adder, represents a way to introduce an additional delay which can be exploited in synthesis for introducing **multicycles**. Also in this case 1 extra clock cycle has been considered sufficient for obtaining a reasonable result in terms of **slack**. Then here is the portion of code regarding the eventual overcoming of the input thresholds for all of the possible frame errors:

```
-- thresholds evaluation
if EN_TH_tot = '1' then
    if perc_err_tot_reg > unsigned(TH_tot) then
        aboveTH_tot <= '1';
    else
        aboveTH_tot <= '0';
    end if;
end if;

if EN_TH_prmbl = '1' then
    if perc_err_prmbl_reg > unsigned(TH_prmbl) then
        aboveTH_prmbl <= '1';
    else
```

```
            aboveTH_prmbl <= '0';
        end if;
    end if;

    if EN_TH_fc = '1' then
        if perc_err_fc_reg > unsigned(TH_fc) then
            aboveTH_fc <= '1';
        else
            aboveTH_fc <= '0';
        end if;
    end if;

    if EN_TH_pl = '1' then
        if perc_err_pl_reg > unsigned(TH_pl) then
            aboveTH_pl <= '1';
        else
            aboveTH_pl <= '0';
        end if;
    end if;
```

It can be seen that every percentage is evaluated against its correspondent threshold dependently on the value of the associated enabling signal that has the role to **mask** the "above" signals that may rise. Then regarding the last part the Monitor Kernel architecture, it can be noticed by looking at figure 4.19 that some logic gates are present in the lower part of the scheme and are reported in the more readable figure 4.24.

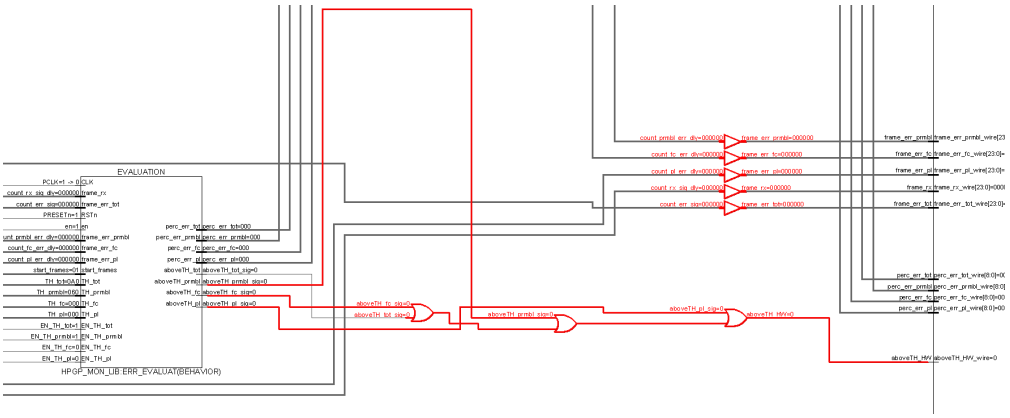

Figure 4.24: View on the or-gates and buffers of the Monitor Kernel architecture

Regarding the buffers they have been put automatically by the simulator to connect the signals that represent the outputs of the counters and those that are the output of the Monitor Kernel while the **or-gates** perform the or logic operation between

the "above" signals in output of the EVALUATION module so that if at least one of them is high, the **aboveTH_HW** signal is asserted that means that an **hardware fail** has been sensed.

4.5.3 HPGP Monitor simulation

At this point there is the need to test the architecture that has been developed in order to implement the **monitoring functionalities** that exploits the HPGP subsystem signals discussed in section 4.4.5. It must be said that in truth many tests have been performed in order to be sure about the correctness of the monitor results and behavior, however for sake of brevity, there will reported just the results of one simulation which represents one of the most complete.

For that simulation it has been considered a sequence of 11 frames where some of them are correct, others are not even received because of a strong applied attenuation and some others have errors in different parts of the frame (preamble, frame control and payload). The frames sequence is recalled in the following figure 4.25.

Figure 4.25: Sequence of the frames involved in the simulation

As it was introduced in section 4.4.5 when dealing with the presentation of the simulation environment, the transmitted frames are read by file by mean of a module named **tb_matrix** whose most meaningful code lines have been treated in that section, while all the values stored in the **Read and Write registers** of the monitor have been defined in the **main.c** file and are reported again for sake of clarity:

```
1  int main()
2  {
3      \\ ...
4      initChip();
5
6      \\ ...
7
8      HPGPMONITOR[0] = 1;          //monitor_en = 1
9      HPGPMONITOR[1] = 1025;       //window = 000000000000000000000100 =
       4   start_frames = 00000001 = 1 --> 2^0 + 2^10 = 1025
10
11     HPGPMONITOR[2] = 160;        //TH_tot = 010100000 --> 0.625
```

```
12      HPGPMONITOR[3] = 96;          //TH_prmbl = 001100000 --> 0.375
13      HPGPMONITOR[4] = 0;           //TH_fc = 000000000
14      HPGPMONITOR[5] = 0;           //TH_pl = 000000000
15
16      HPGPMONITOR[6] = 3;           //EN_TH_tot = 1 and EN_TH_prmbl = 1;
17
18      HPGPMONITOR[7] = 0;           //MUX_sel = 0 --> HW fail sensed,
        SW_fail = 0
19
20      \\ ...
21      return 0;
22  }
```

So now, in figure 4.26, the results of the simulation can be appreciated. As a matter of fact it can be observed that the Monitor IP behaves exactly as expected because it retrieves the values of the percentages in a window of 4 received frames after 1 frame as stated by the values of the signal "start_frames" and "window" stored in their dedicated register, moreover the values of the percentages update on the falling edge of the energy_det signal of the received frames and in a way that is coherent with the sequence of frames. Considering the first 4 received frames, it can be noticed that the first one is affected by a **payload error** but there is not an updating on the percentages because as it has been said, the system works so that it has to wait for a number of frames specified by the value of start_frames (1 in this case) before delivering a meaningful value for the percentages. It is obvious that having a start_frames signal value equal to 1 is not a nice choice but this simulation has just a demonstrative purpose, of course in a real application the Monitor will deal with an higher value and also the considered window of received frames will be larger. Continuing with the simulation, the second frame that has been transmitted is not received since it is affected by a strong attenuation of 40 dB which makes the energy_det signal not to arise, while the subsequent frames have enough energy for being received, however 1 of them presents a **frame control error** that together with the payload one of the first frame makes the total percentage error to be 0.67, above the set threshold of 0.625. This condition makes the **HPGP fail** signal to arise and then fall down when the subsequent frame, which is correct, is received since the same percentage value from 0.67 becomes 0.5 because at that moment there are 2 wrong received frames out of 4 (previously they were 2 out of 3). The rest of the simulation proceeds in a very similar manner, so after the first 4 received frames the counters and the percentage values are reset to 0 and the Monitor starts again its analysis. In this second window, at the time the third frame is received, the percentage of preamble errors has a value of 0.5 which is higher than the respective threshold which is 0.375 so the fail signal arises again. This because both the **enabling signals EN_TOT and EN_PRMBL** for the correspondent thresholds have been enabled in the **main.c** of the simulation.

The **hpgp_fail** signal at the output of the Monitor is directly connected to the **aboveTH_HW** signal because in the C code it has been specified that the selection signal of the MUX (**MUX_sel**) must be 0 and for that value, the output of the MUX is connected to that input while if it was 1, it would be connected to the **SW_fail** signal that in this simulation, in any case, is 0.

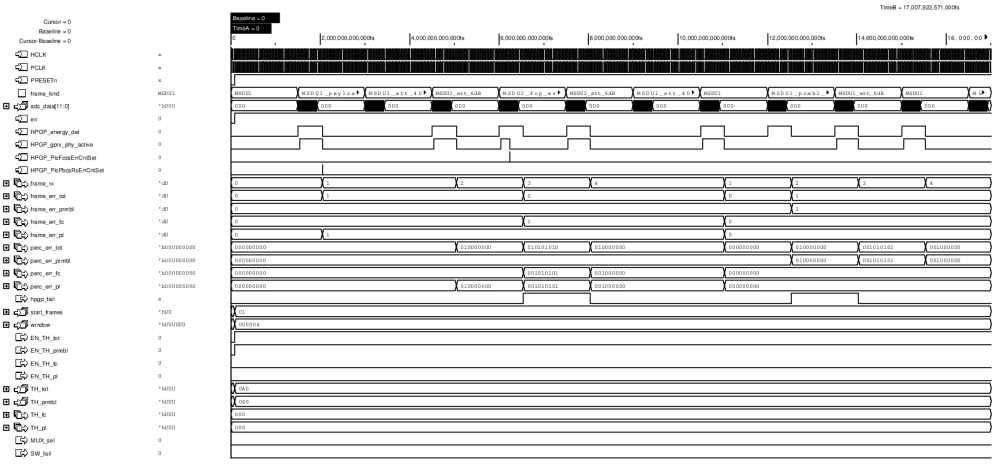

Figure 4.26: Waveforms of the HPGP Monitor simulation

4.5.4 Monitor IP Synthesis and its optimized version

At this point of the Monitor IP analysis, it has been of strong importance to analyze and report some **synthesis results** in order to get some information especially about the dimensions of the IP with respect to the entire project of the EV PLC Modem.

Table 4.2 reports the **occupied area** in μm^2 of the specified digital units as well as their approximated **gate count**. The latter has been obtained through the use of the **Standard Cell Library** of the 40 nm CMOS technology obtained from the **shrinking** of the 45 nm one, taking as a reference the unitary area of the **NAND2** gate with X4 as **drive strength**. In particular the **low-speed** corner of the technology has been used because the reference clock is just 150 MHz. The **cell size** of the chosen gate is specified in table 4.1.

It can be easily denoted that the area occupied by the Monitor IP is a small fraction of the total area of the EV PLC Modem, precisely the 0.43%. This means that it has not a great importance to waste time trying to optimize in the best way the Monitor IP from what concern the occupied area. In addition there is also no need to boost

Drive Strength	**Height (μm)**	**Width (μm)**	**Area (μm^2)**
HS45_LS_NAND2X2_P4	1.960	0.540	1.0584
HS45_LS_NAND2X4_P4	1.960	0.540	1.0584
HS45_LS_NAND2X5_P4	1.960	0.540	1.0584
HS45_LS_NAND2X7_P4	1.960	0.540	1.0584
HS45_LS_NAND2X11_P4	1.960	0.900	1.7640
HS45_LS_NAND2X14_P4	1.960	0.900	1.7640
HS45_LS_NAND2X21_P4	1.960	1.260	2.4696
HS45_LS_NAND2X29_P4	1.960	1.620	3.1752
HS45_LS_NAND2X43_P4	1.960	2.340	4.5864
HS45_LS_NAND2X57_P4	1.960	3.060	5.9976
HS45_LS_NAND2X62_P4	1.960	2.160	4.2336
HS45_LS_NAND2X71_P4	1.960	2.340	4.5864
HS45_LSS_NAND2X64_P4	1.960	3.420	6.7032
HS45_LSS_NAND2X71_P4	1.960	3.780	7.4088

Table 4.1: NAND2 Cell Size

Unit	**Occupied Area (μm^2)**	**Gate count (NAND2X4)**
evplc_top	11698351.297	11052865
.../i_HPGP_monitor	49477.025	46747
.../i_HPGP_monitor_apb_itf	1412.964	1337
.../i_HPGP_monitor_kernel	48045.010	45394
.../i_HPGP_monitor_kernel/EVALUATION	44137.750	41703

Table 4.2: Area report

its performances making its area to increase, because the Monitor must work at 150 MHz and having a version of it that is capable to work at an higher frequency would be useless. Anyway, concentrating on the occupied area of the monitor, it will be discovered that the highest contribution is provided by the **EVALUATION** unit inside the Monitor Kernel which represents alone the 89.2% of the Monitor area.

For this reason an **optimized version** of the Monitor IP has been designed but not implemented in the FPGA due to to time reasons and to what has been said previously, so the small impact on the overall project. In this version the goal was to act directly on the EVALUATION unit, making it more compact to decrease its impact on the Monitor area. In order to do so, it has been thought to remove the parallel computation of 4 divisions and make a single one per clock cycle instead, in order to reuse the same resources and saving area. This new approach led to removal of the 4 inputs to compute the percentages in the EVALUATION

block, leaving just one of them that comes directly from a MUX that selects its output among the several error signals and its selection is managed by a **control unit** which changes its output value at every clock cycle. At the output side of the EVALUATION block, the single computed percentage error value goes in input to a DEMUX that is again controlled by the same control unit, which binds at every clock cycle that generic percentage to a specific one. A schematic that reports the main changes introduced by the optimization of the EVALUATION unit and the Monitor Kernel is reported in figure 4.27 where the changes with respect to the standard architecture are highlighted in yellow.

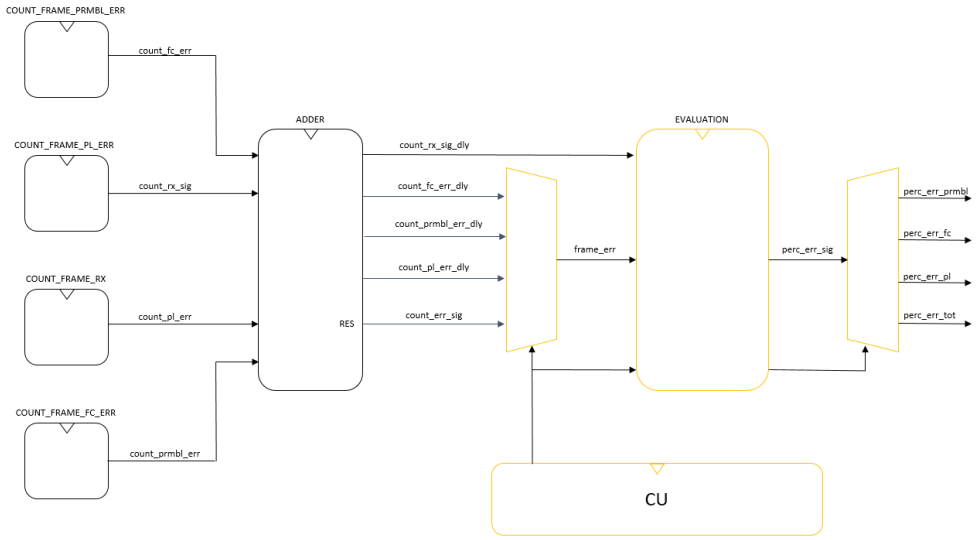

Figure 4.27: High-level schematic of the introduced changes in the Monitor Kernel architecture

In order to appreciate the improvements introduced by this optimization in the EVALUATION unit architecture, a new area report has been obtained from the synthesis and its main results are reported in table 4.3.

Unit	Occupied Area (μm^2)	Gate count (NAND2X4)
evplc_top	11686376.431	11041550
.../i_HPGP_monitor	48120.036	45465
.../i_HPGP_monitor_apb_itf	1388.267	1312
.../i_HPGP_monitor_kernel	46234.463	43684
.../i_HPGP_monitor_kernel/EVALUATION	41246.684	38971

Table 4.3: Area report after the optimization

At this point, making the same calculations, it is immediate to denote that the

impact of the Monitor IP over the whole chip is decreased to 0.41% and the impact of the EVALUATION unit over the Monitor area has changed from 89.2% to 85.7%. As it was expected, there is not such a big impact of the modifications over the top level unit. For this reason there is no need that this new version will be implemented in the future releases since there is just a small improvement in terms of occupied area relative to an IP that plays a very small role in the usage of resources of the entire project.

4.5.5 Observations and future improvements

The proposed architecture for the HPGP Monitor IP it is just the first version of something that will be improved and extended until the **tape-out** of the entire Modem chip. As a matter of fact there are some points that can be considered as crucial for a future refinement.

First of all it could be possible to extend the functionalities of the **EVALUATION** block in the Monitor Kernel. Actually it is quite simple since it just calculates the percentage values of each kind of error on the number of the received frames and recognizes the overcoming of the several thresholds that has been set, however that was just a first basic feature that has been implemented. As a matter of fact, there could be introduced also **absolute value thresholds** or a logic that recognizes if a **certain sequence of wrong frames** has been received. These are just examples, however decisions about that topic will eventually be taken in future when there will be more time to study real streams of frames during a communication, so that the team can be aware of what are those particular conditions that make all the communication to be failed. As a matter of fact, what has been developed until now is an IP with generic monitoring capabilities that has to be refined on the basis of a real conditions behavior that can be done just at **FPGA level** before the realization of the Modem IP.

Another possible feature to implement in the Monitor could be an **energy detection unit**. It may be considered useless since the Monitor receives all the information that needs about energy by exploiting the energy_det signal form the AGC that is an indicator of the overcoming of the EN_TH threshold under which the HPGP subsystem does not consider a frame as received. From a simulation point of view, as it can be seen by looking at figure 4.26, it can be observed that between a frame and another one there is no data on the power line. This is true for the simulation, however in a real case there will be always some **noise** on the power line, for this reason although the HPGP subsystem does not consider as received any signal under a certain power level, there may be the will to make a distinction between this **power line noise** and a **strongly attenuated frame**. The idea is

to tune a **new threshold** value under which the signal on the power line can be considered noise and between it and the already defined threshold EN_TH in the AGC can be considered just as strongly attenuated, in order to have a further way to identify the information that is exchanged in the power line. That feature could also be implemented by using the hpgp_rssi_ma[7:0] signal from the AGC that has been already described, however it could be better to develop a brand new unit in order to calculate the **absolute power** of the received samples in dB, in order to have a direct relationship with the values of power provided by the oscilloscope in the laboratory that are needed as reference to set the values of the thresholds.

The last important aspect that may be addressed is the **granularity of the percentage values**. As a matter of fact, as it has been said in section 4.5.2, the 9-bit fixed point format that has been used allows for a precision of 0.39% for the percentage values which considering a real charging application case could be not sufficient, because there may be millions of received frames and just very few that are corrupted (ideally zero). This situation could potentially lead to select a very large window, using all the 24 available bits for instance, having possibly 1 or 2 wrong frames out of $2^{24} - 1 = 16777215$ obtaining a mathematical percentage value that is several orders of magnitude lower than 0.39% and for this reason not possible to monitor with precision.
The most important thing to say about that topic is that this is a Monitor IP and its main target is to provide a fail signal after a certain threshold is exceeded, not to **provide an accurate statistics** computed over the monitored signals. This means that it is not important to sense precisely the percentage errors if the purpose is to establish if raising the fail signal or not because reasonably, if something goes wrong in the communication, many frames will be affected by some kind of corruption and so an **infinitesimal precision** in the computations will not be needed.
Anyway, improving the Monitor IP in such a way to make it more precise in calculations could be a possibility that will eventually be discussed in the team in the future.

4.6 FPGA validation in laboratory

The last activity that has been performed regarding the development of the Monitor IP has been to **implement** and **validate** it on the FPGA of the lab whose model is **Xilinx Ultrascale XCKU040**.

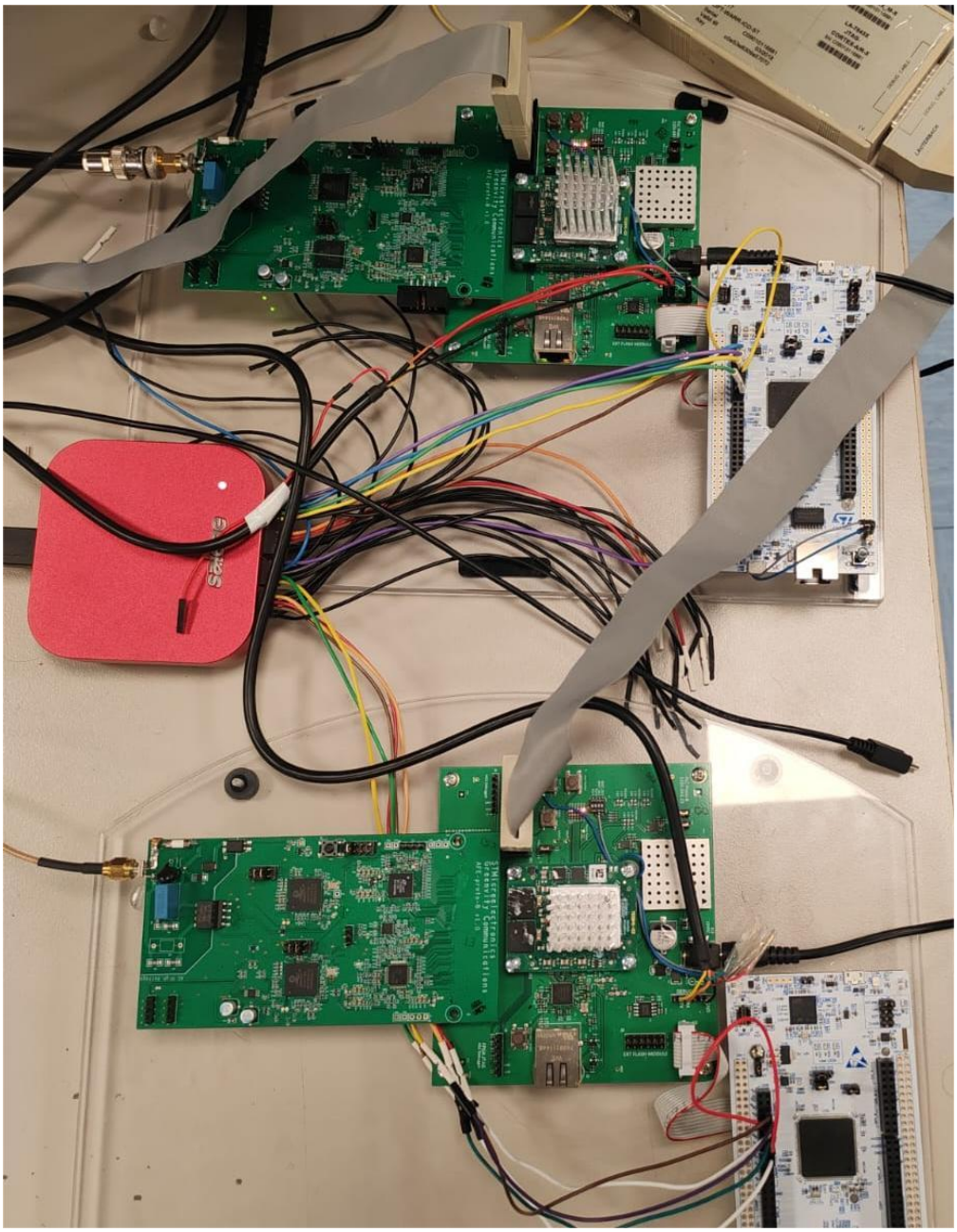

Figure 4.28: Photo of the FPGA testing environment

The **testing environment** that has been set up is the one reported in figure 4.28 where there can be distinguished two identical FPGAs that implement the whole EV PLC Modem with the lower one that acts as a transmitter and the upper

one as a receiver. In order to be precise, each device consists in an FPGA that implement the Baseband of the HPGP subsystem, the RISC-V and the rest of the digital IPs while the analog components such as the HPGP AFE have been realized in a dedicated board that is the one with its long side set in horizontal. Both the devices are connected to two identical **STM32 Host MCUs** which have the role to provide the information to be exchanged via **SPI** to the respective EV PLC Modem that at their turn will elaborate these information in HPGP frames exchanged through the power-line (the cable connected to the AFE of the analog board of the transmitter FPGA at the bottom left of figure 4.28) as described in chapter 2. In addition, the two MCUs set properly the most important **network parameters** discussed in chapter 2 of the HPGP communication that are the **SNID** that defines unambiguously the network which has to be the same for both the devices, the **STEI** and the **DTEI** which are respectively the identifiers of the station that is transmitting and receiving. In the lab case, these values for the three identifiers has been used:

- FPGA-1 (TX): SNID=0xB, STEI=0x2, DTEI=0x3
- FPGA-2 (RX): SNID=0xB, STEI=0x3, DTEI=0x2

It can be noticed that STEI and DTEI are reversed in the two EV PLC Modems implemented in the FPGAs because the destination identifier of the transmitter must coincide with the source identifier of the receiver and viceversa since in HPGP communication the receiver has been thought for sending back an acknowledge frame to the transmitter if the communication did not failed.
The testing environment is characterised also by the presence of an **oscilloscope** that is connected to the power-line and placed between the two modems so that the HPGP frames can be observed on its screen as it is showed in figure 4.29 where there can be seen an HPGP frame and its correspondent acknowledge.
Then the last device which completes the laboratory set is a **TRACE32 Lauterbach Debugger** that is connected to the Modem processor by mean of its debug port and it is very helpful since it gives access to a software environment from which the user is able to write values in registers for configuring properly the device and it can perform at a constant frequency a load of the overall memory map registers so that their content becomes visible through its GUI. Figure 4.30 reports the picture of the Lauterbach Debugger in the ST lab.

The first test that has been performed consisted simply in **counting** the received frames, just for having an initial idea about the correctness of the basic monitor functionalities. In order to do that a correct version of a MSDU1 frame has been loaded on the oscilloscope to be transmitted as many times as wanted to the RX Modem. In this test and in the following that will be described, the frames are

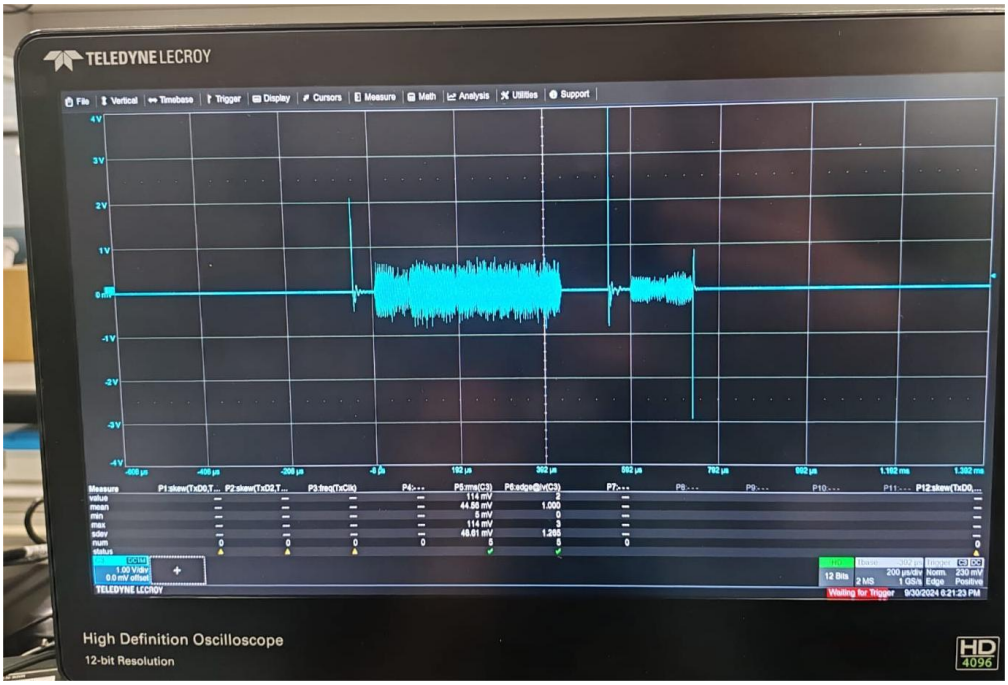

Figure 4.29: HPGP frame on the power-line and its acknowledge seen from the oscilloscope

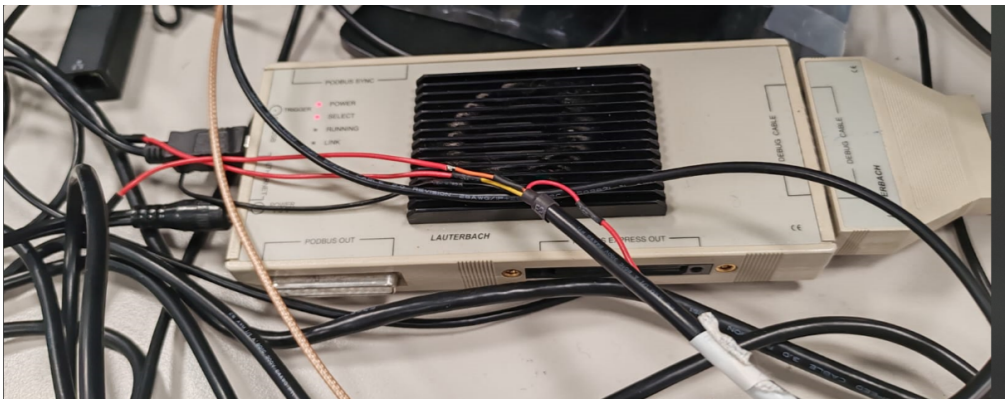

Figure 4.30: TRACE32 Lauterbach Debugger

transmitted via oscilloscope and then received by the RX EV PLC Modem without making them to be generated by the other TX Modem. This because the main purpose of this tests is to verify the correct functionality of the Monitor and so sending them through the oscilloscope allows to send also the frames with the corrupted samples obtained in Matlab as explained before while generating the frames by mean of the TX Modem does not allow to introduce errors for testing

purposes where desired.

After the preparation of this setting what has been expected by the observation of the values stored in the Monitor IP registers through the use of the Lauterbach Debugger was the totality of the transmitted frames to be received without any error.

In practice, the obtained result was slightly different as it can be seen by the observation of the registers values on the Debugger after having transmitted just one MSDU1 frame.

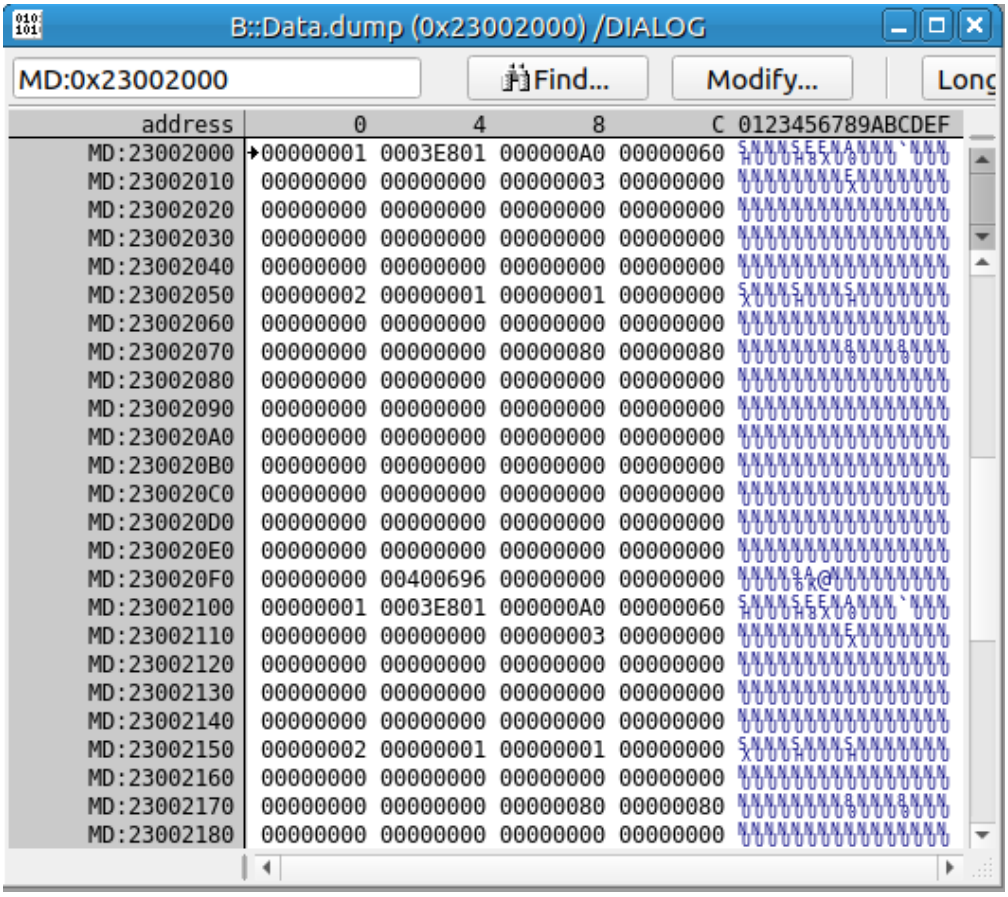

Figure 4.31: Monitor IP registers seen through the Lauterbach Debugger after the transmission of one frame

Knowing that the register at the address 0x23002050 stores the value of the **received frames** and those at the addresses 0x23002054, 0x23002058, 0x2300205C and 0x23002060 respectively store the amount of **total errors**, **preamble errors**, **frame control errors** and **payload errors**, it results that the received frames are

actually 2 instead of 1 and that the extra received frame seems to be a preamble error. The explanation of that phenomenon is due to the fact that the FPGA prototype still suffers from **non-idealities** and when it is enabled its AFE board generates a spike at high energy on the power line that is read by the Monitor IP as a frame at a sufficient energy to be seen but without a recognizable preamble. In figure 4.32 there can be visualized the spike on the oscilloscope and in figure 4.33 is showed that, when enabled, the receiver interprets that spike as a received frame wrong in preamble.

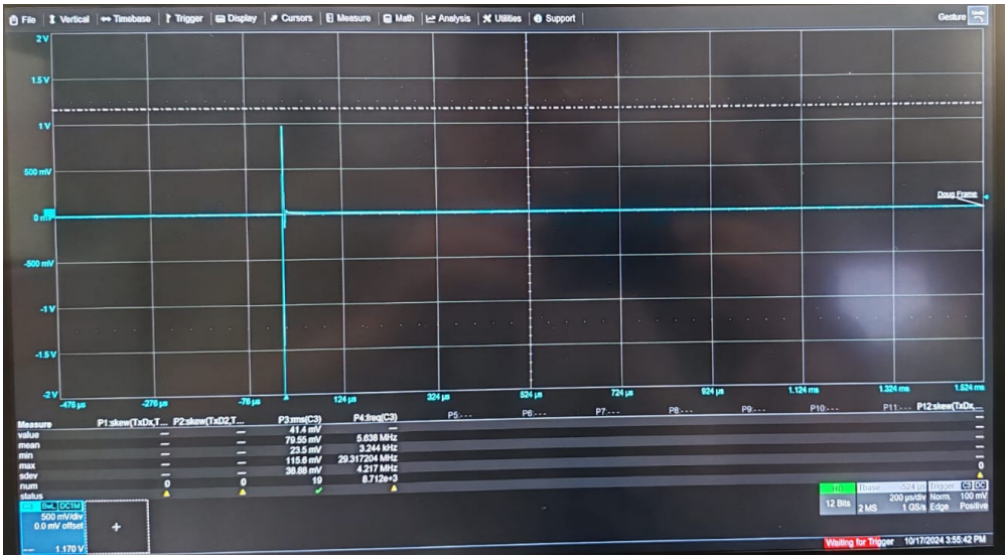

Figure 4.32: Spike revealed by the oscilloscope that is interpreted as a wrong frame by the Monitor IP

After that very initial test other 5 frames have been transmitted. This time, as it can be checked in figure 4.34, the received frames are 8 instead of the expected 7, this because again another spike appeared during the transmission as can be seen by looking at the value stored in the register at address 0x23002058 and more in general in the one of the total received wrong frames at address 0x23002054.
The other meaningful test to be done has been to replicate on the lab prototype the same test that has been performed with the RTL waveforms, so the same sequence of frames in figure 4.25 has been loaded on the oscilloscope as reported in figure 4.35.
Since the presence of the spike has not been yet resolved and it is stochastic, this time it affected one of the correct frames on its frame control portion, making it appear to be a corrupted frame. The registers values can be observed in figure 4.36.
So there can be seen that 8 out of 11 frames has been received because the energy

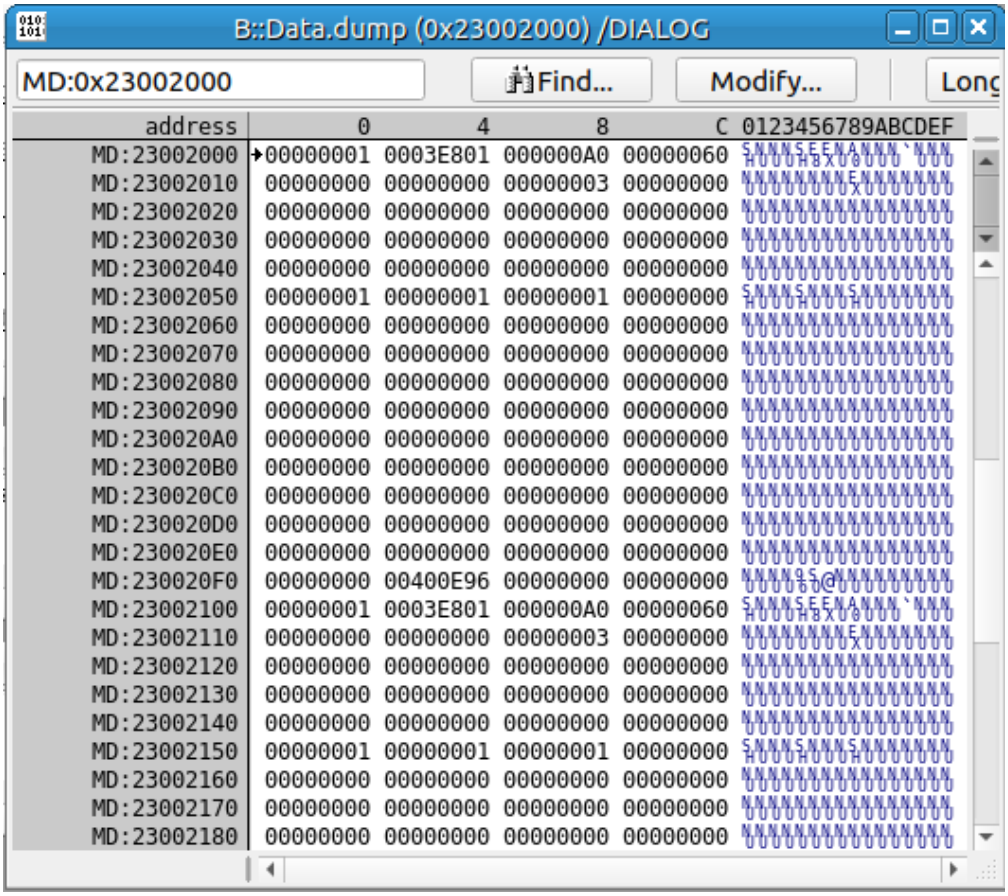

Figure 4.33: Monitor IP registers seen through the Lauterbach Debugger after the enabling of the device

of the 3 frames attenuated of 40 dB has not overcome the energy threshold. In addition all the errors on the three different frame portions have been sensed, with one more error related to the frame control due to the spike as said before. This situation led to a total error rate of $4/8 = 0.5$ which is correctly reported in the **PERC_ERR_TOT_REG** at address 0x23002078 in 9-bit fixed-point format as 00000080. As an example, a further check can be done by looking at register value at address 0x2300207C which is 00000020 since it is representative of the preamble error ratio in 9-bit fixed-point format.

These simple tests proved that the Monitor IP works in a correct and controllable way. Anyway, it has to be said that what has been done until now is just a **functional test** of the Monitor IP features and not a **real case test** in which a large amount of frames with characteristics correspondent to the ones of a **real**

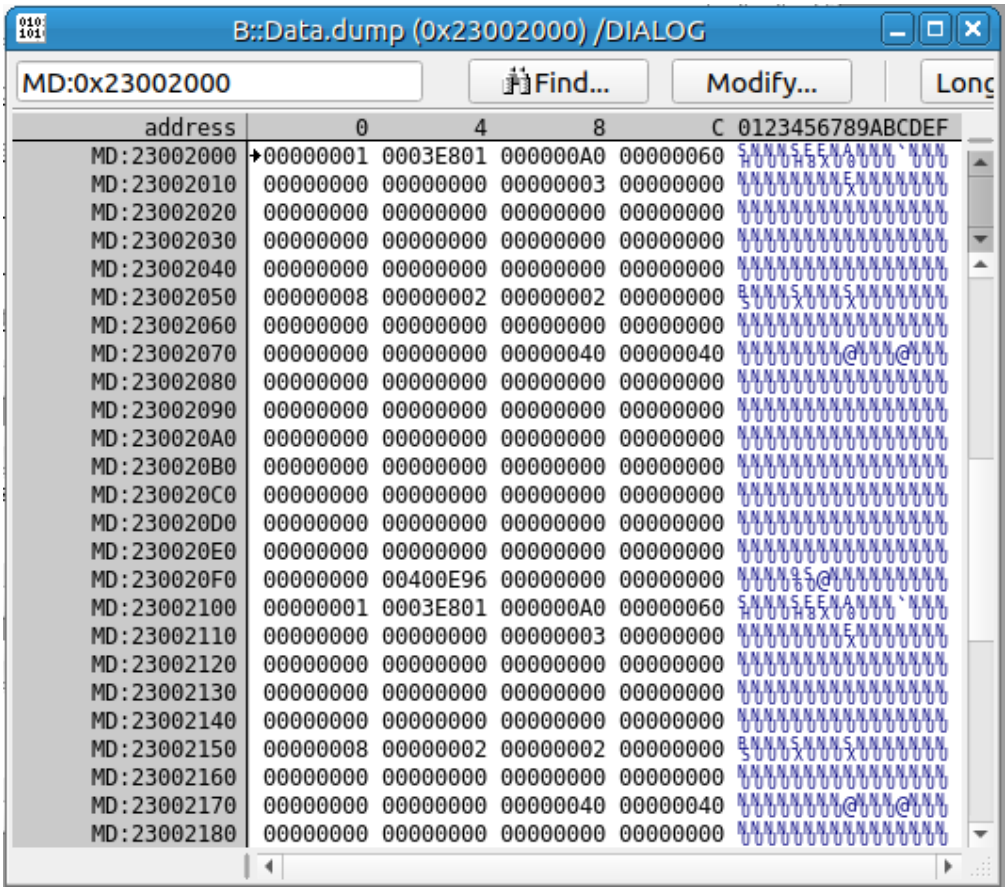

Figure 4.34: Monitor IP registers seen through the Lauterbach Debugger after the transmission of other 5 frames

charging operation are exchanged between the two device. This last test still has to be performed and it will be done in the next months.

Figure 4.35: Frame sequence transmitted to the RX Modem correspondent to the test case considered in figure 4.25

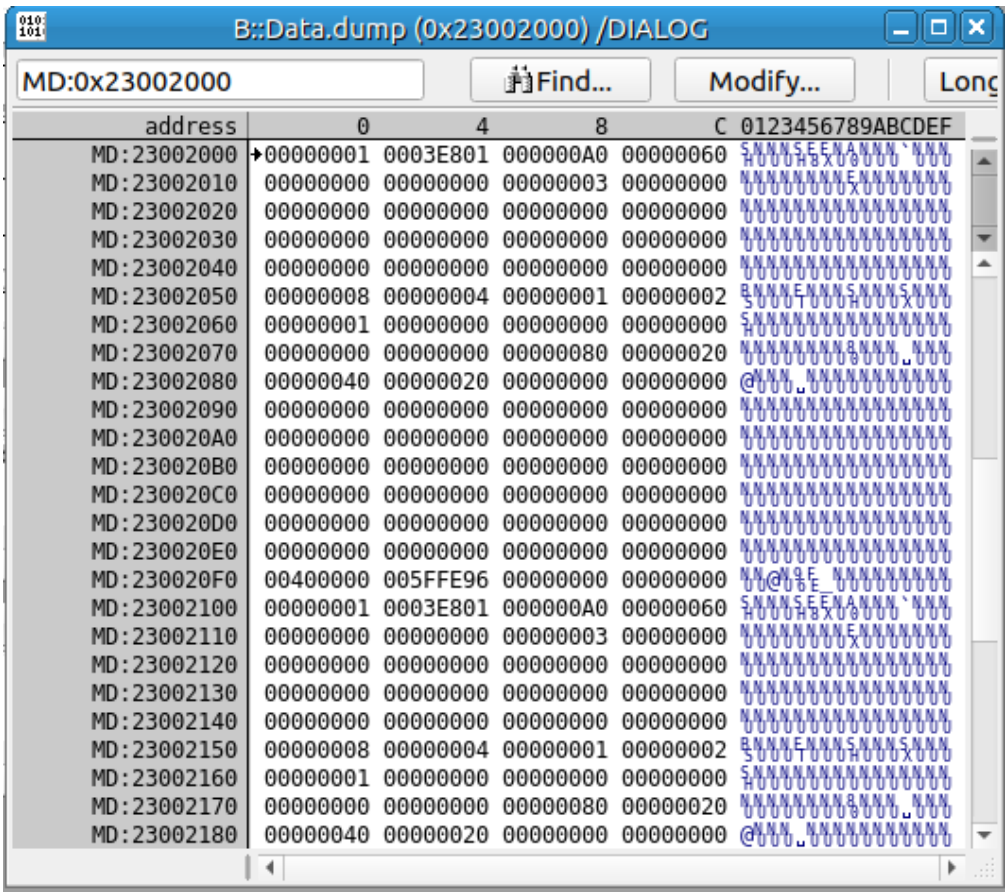
B::Data.dump (0x23002000) /DIALOG
MD:0x23002000
Find...
Modify...
Long
address 0 4 8 C 0123456789ABCDEF
MD:23002000 00000001 0003E801 000000A0 00000060
MD:23002010 00000000 00000000 00000003 00000000
MD:23002020 00000000 00000000 00000000 00000000
MD:23002030 00000000 00000000 00000000 00000000
MD:23002040 00000000 00000000 00000000 00000000
MD:23002050 00000008 00000004 00000001 00000002
MD:23002060 00000001 00000000 00000000 00000000
MD:23002070 00000000 00000000 00000080 00000020
MD:23002080 00000040 00000020 00000000 00000000
MD:23002090 00000000 00000000 00000000 00000000
MD:230020A0 00000000 00000000 00000000 00000000
MD:230020B0 00000000 00000000 00000000 00000000
MD:230020C0 00000000 00000000 00000000 00000000
MD:230020D0 00000000 00000000 00000000 00000000
MD:230020E0 00000000 00000000 00000000 00000000
MD:230020F0 00400000 005FFE96 00000000 00000000
MD:23002100 00000001 0003E801 000000A0 00000060
MD:23002110 00000000 00000000 00000003 00000000
MD:23002120 00000000 00000000 00000000 00000000
MD:23002130 00000000 00000000 00000000 00000000
MD:23002140 00000000 00000000 00000000 00000000
MD:23002150 00000008 00000004 00000001 00000002
MD:23002160 00000001 00000000 00000000 00000000
MD:23002170 00000000 00000000 00000080 00000020
MD:23002180 00000040 00000020 00000000 00000000

Figure 4.36

Chapter 5

Conclusions

In conclusion, what has been exposed in this thesis is the whole process that started from the **recognition of a deficiency** in terms of **Functional Safety** of an almost entirely developed large project as the **EV PLC Modem**, and led to the **implementation on an FPGA prototype** of the latter, with the integration of a digital hardware IP as the described **Monitor** that actually represent the solution to the initial safety issue and completes the project.

What it is important to be highlighted at this point, is that the project is not still near to its **tape out** so to the realization of the EV PLC Modem chip on the silicon. This mostly for the fact that there are still issues regarding its prototyping on the FPGA, such as the already mentioned spikes produced by the Modem in RX mode over the power line and the not perfect functioning of Ethernet port. These issues will be responsible for further refinements and so there will be other **releases** of the Modem that eventually would also give the possibility to make modifications on the RTL architecture of the Monitor IP following the ideas mentioned in section 4.5.5.

Another key aspect will be to set up a **real case test** with the FPGA prototypes in the lab in which the TX Modem should be able to transmit on the power line a long set of frames that has to emulate the one that is exchanged between an EV and an EVSE during a charging. This last test will give the possibility to make a better tuning of the **parameters** of the Monitor IP, such as its threshold values and the frame window taken into consideration for the computation of the statistics.

Bibliography

[1] Council of the European Union. *Clean and sustainable mobility*. 2024. URL: https://www.consilium.europa.eu/en/policies/clean-and-sustainable-mobility/ (cit. on pp. 1, 2).

[2] European Commission. *Smart grids and meters*. 2024. URL: https://energy.ec.europa.eu/topics/markets-and-consumers/smart-grids-and-meters_en (cit. on p. 2).

[3] Nishant Sagar. «Powerline Communication Systems: overview and analysis». MA thesis. New Jersey: Rutgers - New Brunswick, 2011 (cit. on p. 2).

[4] *HomePlug Green PHY Specification*. Powerline Alliance, 2012 (cit. on pp. 3, 5, 13, 16, 20, 25, 34, 41, 42, 77).

[5] H. A. Latchman. *HomePlug AV and IEEE 1901: a Handbook for PLC Designers and Users*. Piscataway, NJ: IEEE Press, 2013 (cit. on pp. 5, 11, 14, 15).

[6] HomePlug Powerline Alliance. *Why use HomePlug*. 2014. URL: https://web.archive.org/web/20140622095946/http://www.homeplug.org/explore-homeplug/overview/ (cit. on pp. 5, 6).

[7] *HomePlug Green PHY: The Standard For In-Home Smart Grid Powerline Communications*. Powerline Alliance, 2010 (cit. on pp. 7, 11–13).

[8] IEEE Erico Guizzo. *Closing in on the perfect code*. 2004. URL: https://spectrum.ieee.org/turbo-codes (cit. on p. 8).

[9] ADAS and Infotainment for Automotive team (ST Microelectronics). «EV PLC Architecture Device Definition». In: (June 2024), p. 6 (cit. on pp. 28, 31, 43, 47, 81).

[10] *ISO 26262: Automotive Functional Safety Standard White Paper*. Internation Organization for Standardization (ISO), 2011 (cit. on p. 53).