Universidade de Aveiro
Departamento de
Electrónica, Telecomunicações e Informática
2014

Cristóvão André
Antunes Cruz

Mecanismos de Suporte para MAC 802.11p Determinística

Support Mechanisms for Deterministic 802.11p MAC

"Don't talk so much, old sport, Play!"

F. Scott Fitzgerald
The Great Gatsby

Cristóvão André Antunes Cruz

Support Mechanisms for Deterministic MAC in IEEE 802.11p

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e de Telecomunicações, realizada sob a orientação científica do Professor Doutor Arnaldo Silva Rodrigues de Oliveira, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Professor Doutor José Joaquim Campos Ferreira, Professor Adjunto da Escola Superior de Tecnologia e Gestão de Águeda da Universidade de Aveiro

o júri / the jury

presidente / president	**Prof. Doutor José Luis Costa Pinto de Azevedo** Professor Auxiliar da Universidade de Aveiro
vogais / examiners committee	**Prof. Doutor Arnaldo Silva Rodrigues de Oliveira** Professor Auxiliar da Universidade de Aveiro (orientador)
	Doutor Paulo Jorge de Campos Bartolomeu Investigador Doutorado da Globaltronic, Electrónica e Telecomunicações, SA

agradecimentos / acknowledgements

Agradeço aos meus orientadores Arnaldo Oliveira e Joaquim Ferreira, não só pela orientação prestada mas também pela enorme dose de paciência que demonstraram perante os meus métodos pouco ortodoxos e prazos falhados.

Agradeço ao grupo de trabalho no seio do qual este projeto foi desenvolvido, nomeadamente ao Bruno Silva, João Almeida, Nelson Cardoso, Sikandar Khan, Manuel Santos, João Pio e Manuel Ventura. Este trabalho só foi possível devido às contribuições de todos eles.

Agradeço à minha família por todo o apoio que prestaram ao longo do meu percurso académico.

Agradeço à Vanessa por me ter incentivado a terminar aquilo que comecei.

Obrigado.

Palavras-chave VANET, Communicações Veículares, ITS, DSRC, IEEE 802.11, SDR, MAC, Tempo-Real

Resumo

Os meios de transporte têm um papel preponderante na sociedade moderna. Muito esforço de investigação tem sido dedicada e este campo nos últimos anos, com vista a tornar estes meios mais limpos, seguros e eficientes, originado os chamados Sistemas de Transporte Inteligentes (ITS). Enquanto algumas das tecnologias base estão já prontas a ser utilizadas, ainda existem problemas a ser resolvidos antes de se poder utilizar todo o potencial destes sistemas. Um problema específico, o acesso ao meio, é atualmente considerado um dos mais desafiantes em termos de investigação e que detém ainda uma quantidade interessante de problemas a ser resolvidos para que se possa depender de comunicações sem fios em ambientes veiculares. Estudos provaram que os protocolos standard não resolvem este problema e têm sido propostas soluções, Mas apesar da dificuldade de modelar corretamente a dinâmica do canal, a maior parte das análises tem sido realizada em ambientes de simulação, com assunções simplistas que não correspondem necessariamente ao ambiente real.

A implementação de um mecanismo de acesso ao meio determinístico é dificultada pelo facto de que os dispositivos comerciais não permitem modificações aos mecanismos standard e o desenvolvimento de um dispositivo de raiz que implemente o mecanismo proposto ser extremamente trabalhoso. No entanto, ao longo dos últimos anos tem sido desenvolvida a plataforma de comunicações veiculares IT2S, que atingiu agora uma fase que permite a sua utilização para implementar e testar novas soluções para ambientes veiculares. Trata-se de uma plataforma flexível que permite a realização de modificações em qualquer camada da pilha protocolar, portanto passível de ser adaptada para a implementação de novos mecanismos de acesso ao meio. Este trabalho apresenta uma perspetiva alargada dos mecanismos de acesso ao meio determinísticos propostos na literatura e estuda quais as características necessárias que um dispositivo de comunicação precisa fornecer para os poder implementar. Segue-se então uma proposta de implementação de um tal dispositivo, baseada na plataforma IT2S.

Foi possível obter uma solução flexível o suficiente para implementar todos os mecanismos estudados recorrendo apenas a *software*, sem necessidade de alterações ao *hardware*, baixando a fasquia da dificuldade na criação de uma implementação prática de um mecanismo de acesso ao meio. A solução foi testada e a desempenho considerada adequada para as possíveis utilizações. É agora possível criar bancadas de teste para novos mecanismos de acesso ao meio e executar análises mais concretas e precisas da sua desempenho.

Keywords VANET, Vehicular Communications, ITS, DSRC, IEEE 802.11, SDR, MAC, Deterministic MAC

Abstract

Transportation systems play an extremely important role in modern society. A huge research effort has been devoted to this field in the past few years making them safer cleaner and more efficient, originating the so-called Intelligent Transportation Systems (ITS). While some of the enabling technologies are entering their mature phase, there are still many open problems that must be solved before such systems can be effectively leveraged. Specifically, the medium access is regarded as being one of the most challenging issues to solve in order to provide dependable wireless communications in vehicular networks [BUSB09]. The standard protocols have been shown to fail in addressing this issue and some possible solutions are being proposed, but despite the difficulty of correctly modelling the channel dynamics, most work on MAC protocols for real-time vehicular communications has been performed under simulated environments, using simplistic assumptions that do not necessarily hold in a read environment.

The implementation of a deterministic MAC scheme is hampered by the fact that commercial devices do not allow modifications to the standard MAC mechanism, and the development of a device from scratch to implement one Medium Access Control (MAC) scheme is an extremely laborious endeavour. However, over the last few years, the IT2S platform for vehicular communication has been developed and is now in a stage that allows implementation and testing of new solutions for the vehicular communications environment. It is a flexible platform that allows modification to be made in any layer of the communication stack and therefore suited to be adapted for the implementation of new MAC schemes. This work presents an overview of MAC mechanisms capable of providing deterministic real-time access and assesses the features a communications device must include in order to allow the implementation of these mechanisms. It then proposes an implementation of such as device based on the existing IT2S plaform.

A flexible solution was obtained that allows all the studied MAC schemes to be implemented purely in software, with no modifications required to the hardware mechanisms, lowering the needed amount of skills required to perform a working implementation of a novel MAC scheme. The performance of the solution was also found to be appropriate for the required uses. It is now possible to create test beds for new MAC schemes and perform more concrete and accurate analysis of their performance.

Contents

List of Figures

List of Tables

Acronyms

AD/DA Analog-to-Digital/Digital-to-Analog.
ADC Analog-to-Digital Converter.
AIS Abbreviation Injury Scale.
API Application Programming Interface.

BSS Basic Service Set.

C2X Car to X.
CA Collision Avoidance.
CCA Clear Channel Assessment.
COTS Commercial Off-The-Shelf.
CSMA/CA Carrier Sense Multiple Access with Collision Avoidance.

DAC Digital-to-Analog Converter.
DCC Decentralized Congestion Control.
DDR Double Data Rate.

EC Elementary Cycle.
EDCA Enhanced Distributed Channel Access.
EIFS extended interframe space.

FCS Frame Check Sequence.
FIFO First In First Out.
FPGA Field Programmable Gate Array.
FSMD finite state machine and datapath.

GPIO General-Purpose Input/Output.
GPS Global Positioning System.

I^{2}C Inter-Integrated Circuit.
IEEE Institute of Electrical and Electronics Engineers.
IT2S IT2S.
ITS Intelligent Transportation Systems.

LMAC Lower MAC.
LNA Low Noise Amplifier.
LOS line of sight.

MAC Medium Access Control.

OBU On Board Unit.
OCB Outside the Context of a BSS.
OFDM Orthogonal Frequency Division Multiplexing.

PA Power Amplifier.
PC Personal Computer.
PER packet error rate.
PHY Physical Layer.
PLB Processor Local Bus.
PLCP Physical Layer Convergence Procedure.
PMD Physical Medium Dependent.
PPS Pulse Per Second.

RF Radio Frequency.
RSSI Received Signal Strength Indication.
RSU Road Side Unit.
RTC real time clock.

SDR Software Defined Radio.
SIFS short interframe space.
SPI Serial Peripheral Interface.
STDMA self-organizing time division multiple access.

TDMA Time Division Multiple Access.

UART Universal Asynchronous Receiver/Transmitter.
UMAC Upper MAC.
USB Universal Serial Bus.

V-FTT vehicular flexible time-triggered.
VANET Vehicular Ad-hoc NETwork.

WiFi Wireless Fidelity.

Chapter 1

Introduction

1.1 Background

"Road traffic injuries are the eight leading cause of death and the leading cause off death for young people aged 15-29. More than a million people die each year on the world's roads, and the cost of dealing with the consequences of these road traffic crashes runs to billions of dollars. Current trends suggest that road traffic deaths will become the fifth leading cause of death unless urgent action is taken." [Org13]

While cars have been including more and more technologies to help prevent accidents, the human driver still controls most of the vehicle trajectory and its own perception limitations are still an important factor in automotive crashes. The limitations are inherent to the human senses and reaction times, which are not well suited for some scenarios, mainly where there is no line-of-sight to the hazard with adequate distance [HG11].

There is however an emerging technology which holds the promise to greatly improve car safety and provide vast potential to the deployment of efficiency and entertainment services on board of the vehicles: Car to X (C2X). Heavily influenced by the IEEE 802.11, this communication technology aims to extend the *time-horizon* of the information related to safety and efficiency that is provided to the driver, complementing the already vast array of sensors at his disposal [PLFE+09]. Boasting access to dedicated wireless spectrum and specially designed communication devices which are capable of operating in the challenging vehicular environment, with high relative speeds and dynamic scenarios, this technology also holds extensive support from government and efforts have been made to provide a common technological framework that allows interoperation between different makers and countries. Much research has been done on this subject, under the guise of cooperative Intelligent Transportation Systems (ITS) and it is now accepted that by having the vehicles communicate with each other and with the infrastructure it is possible to prevent accidents and improve traffic efficiency in ways that were not possible before [MTC+10].

1.2 Purpose

Despite the use of dedicated spectrum and specially designed devices, the current standards governing Vehicular Ad-hoc NETworks (VANETs) still fall short of providing dependable MAC, as is desirable if this technology is to be used in support of safety applications, where danger to human life is involved [BUSB09]. On top of that, study of novel MAC mech-

anisms is hampered by the fact that the highly dynamic channel makes it difficult to properly conduct simulated experiments [ESG13], calling for the use of actual devices that implement the relevant standards but at the same time allow experimentation with new mechanisms that may conflict with existing regulations. Such device is currently not available and most research on this field is conducted in simulation environments, despite the already identified issues, leading to analysis that ignore problems such as hidden and exposed nodes, or even the presence of alien MAC schemes, therefore failing to completely address the problem at hand. There is a pressing need for a device that allows the experimentation of new ideas related to MAC in C2X [COFM14].

1.3 Objectives

The main objective of this work is to create a device flexible enough to provide access to the basic mechanisms that allow implementation of various deterministic MAC methods. In order to do so, an overview of the literature is going to be conducted, in order to assess the required mechanisms. It will then present the IT2S (IT2S) device, a standards compliant vehicular communication platform that has been developed over the last few years under the Headway [Hea14] and ICSI [ICS14] projects. This device takes advantage of reconfigurable hardware technology and follows a Software Defined Radio (SDR) approach, making it very flexible to modifications in order to cope with new requirements. Based on the identified requirements and the IT2S platform capabilities, a new device will be proposed, implemented and tested. The main objectives of this work are then:

- Assess the basic mechanisms required for the implementation of deterministic MAC methods;
- Propose modifications to the IT2S device to implement the assessed mechanisms;
- Implement the proposed modifications;
- Evaluate the performance of the solution.

1.4 Structure

This dissertation continues with 6 more chapters organized as follows:

- **Chapter 2 - Fundamental Concepts** — Summary of the standard MAC capabilities and limitations followed by a study of the deterministic MAC mechanisms, specifically designed for VANETs, currently found in the literature, as well as which hardware capabilities are needed in order to implement such mechanisms;
- **Chapter 3 - The IT2S Experimental Vehicular Communication Platform** — Detailed description of the IT2S platform, on which the implementation will be based, including the hardware and all the relevant operational details;
- **Chapter 4 - Architecture** — Proposal of a specification for a device capable of implementing the studied deterministic MAC schemes, including a detailed description of the necessary modifications to the IT2S platform;

- **Chapter 5 - Implementation** — Description of the executed implementation;
- **Chapter 6 - Validation** — Presentation of the test results and assessment of the adequacy of the implemented device for the testing of deterministic MAC mechanisms;
- **Chapter 7 - Conclusions and Future Work** — Final remarks and identification of future work directions.

Chapter 2

Fundamental Concepts

The field of ITS has been intensively researched over the last few years with the intention of providing enhanced sensing capabilities to vehicles in order to enhance both safety and efficiency [PLFE+09]. By providing mechanisms for vehicles to communicate between each other, this technology gives a foundation for many applications such as: intelligent intersections, traffic congestion control, accident detection and warning and platooning, too name a few. However, for many of those applications to be effective, a common technological foundation and sufficient market penetration is required. In order to obtain those, a mix of standardization and government commitment effort has been developed, leading to the approval of several standards that govern many aspects of this technology and allocation of dedicated wireless spectrum, achieving what could be considered a wide geographical interoperability [MTC+10]. But even with all this effort, there are still some problems to solve in order to make this technology dependable enough to be used in life threatening situations such as deterministic MAC [COFM14].

This chapter starts by presenting the currently relevant technological standards. It then follows to identify the more pressing shortcomings and studies the state of the art of how to overcome them.

2.1 Vehicular Communications Standards

2.1.1 IEEE 802.11p

The physical layer used in vehicular communication devices is based on the IEEE 802.11 standard. This standard, widely known as Wireless Fidelity (WiFi) generally used by personal computers worldwide, governs not only the physical properties of the radio communication but also the MAC mechanisms that must be used to provide fair access to the communication channel. Support for vehicular environments was introduced in amendment p and finally included in the 2012 revision of the standard, IEEE 802.11-2012.

At the physical level, this amendment is heavily based on the previously existing a amendment, using the same modulation scheme. However, the channel bandwidth was reduced in half in order to address both the Doppler effect caused caused by the high mobility of the nodes and the inter-symbol interference caused by the reflections and multipath propagation that are common in the vehicular environments. Other specifics such as operating frequency and transmission power are not defined by this standard and is governed by the regulatory bodies of each region.

AC	CWmin	CWmax	AIFSN	TXOP Limit OFDM/CCK-OFDM PHY
AC_BK	aCWmin	aCWmax	9	0
AC_BE	aCWmin	aCWmax	6	0
AC_VI	(aCWmin+1)/2–1	aCWmin	3	0
AC_VO	(aCWmin+1)/4–1	(aCWmin+1)/2–1	2	0

Table 2.1: EDCA parameter set for OCB stations[8-106] [IEE12]

The MAC mechanism adopted is based on Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) with the amendment e, Enhanced Distributed Channel Access (EDCA). Two main operations take part in this mechanism: Clear Channel Assessment (CCA) and Collision Avoidance (CA). A channel is considered busy whenever either the Received Signal Strength Indication (RSSI) rises above a predetermined level or the preamble of a packet is detected. On this last case, the channel will remain busy for the declared packet duration. The collision avoidance mechanism works by requiring the medium to be perceived as free for some time before transmission of a packet can start. The amount of time is selected randomly from a range depending on the packet priority. Higher priority packets select a value with a lower maximum value, while the contrary happens for lower priority packets. The specific minimum and maximum values are presented in table 2.1 under the names `CWmin` and `CWmax` respectively, with the values of `aCWmin` and `aCWmax` being defined as 15 and 1023 respectively. The selected number represents the amount of time slots the medium needs to be perceived as free in order to execute a transmission. The duration of each slot is also dependent on the packet priority, with the specific value being $AIFSN * aSlotTime$. The standard defines `aSlotTime` to be 13 µs for the physical layer used in vehicular environments, so, the slot time will effectively vary from 117 µs on the lowest priority frames to 26 µs on the highest priority frames.

The last important feature of this amendment is that communications occurring within the frequencies allocated for vehicular communications must occur Outside the Context of a BSS (OCB). *Regular* 802.11 based communications occur within the context of a Basic Service Set (BSS) and require association procedures that introduce overhead and latencies in the communication process. Given the volatility of vehicular networks it was deemed necessary to bypass all these mechanisms and communicate constantly in broadcast mode. One of the immediate drawbacks of this method is that the MAC layer is agnostic to traffic security and nodes identity, and the implementation of such facilities is left to the application layer.

Despite all the efforts to provide robust MAC and physical layers, studies reached the overall conclusion that under high network loads the medium access converges to an ALOHA like behaviour where each station transmits in a random time slot independently of the neighbouring stations, causing unbounded collisions [SWL+12][BUSB09]. This limitation sparked vast amounts of work concerning possible improvements, some of which will be presented in the next section.

2.1.2 ETSI ITS G5

The ETSI ITS G5 is the european set of standards that govern vehicular communications. It is a wide ranging work that encompasses all layers of the communication stack, from physical device to the types of exchanged messages. The physical layer in use under this standard is the one defined in 802.11p, but the MAC scheme, Decentralized Congestion Control (DCC) is an enhanced version specifically designed to prevent some of the problems identified in the Institute of Electrical and Electronics Engineers (IEEE) standard proposal, namely congestion under high traffic load [ACM+13].

DCC is a congestion control mechanism which works by modifying PHY and MAC parameters in a decentralized fashion in order to try to provide adequate medium access under different channel load conditions. It is composed of several modules that operate on different layers of the network stack:

- *DCC-Facility*[ETSI TS 102 636] controls the messaging rate in the application layer;
- *DCC-Net*[ETSI TS 103 141] provides various priorities to different message types;
- *DCC-Access* [ETSI TS 102 687] allows modification of the physical layer operation parameters;
- *DCC-Management* coordinates all the other components and provides a stable cross-layer mechanism.

Not all of the DCC components are completely defined yet and the work that has been carried out so far mainly concerns the *DCC-Access* module. This module is also of special importance to this dissertation due to the fact that it deals directly with the operation parameters of the physical layer, namely:

- Transmit Power Control (TPC) : transmission power;
- Transmit Rate Control (TRC) : packet transmission rate;
- Transmit Datarate Control (TDC) : transmission bitrate;
- DCC Sensitivity Control (DSC) : CCA parameters;
- Transmit Access Control (TAC) : packet prioritization.

These parameters are changed according to the perceived medium state, as observed in fig. 2.1, and at any given moment the device can be in one of three states: *relaxed*, *active* or *restrictive*. Given the decentralized nature of this mechanism, all decisions are made locally, based solely on the measure channel load, CL in the image, which is defined as the amount of time the signal is above the CCA threshold, or, the channel is declared as busy.

This mechanism has been studied by several author under simulated environments and the general conclusion seems to be that DCC is also not suited to cope with high network loads [SWL+12][ACM+13][ESG13].

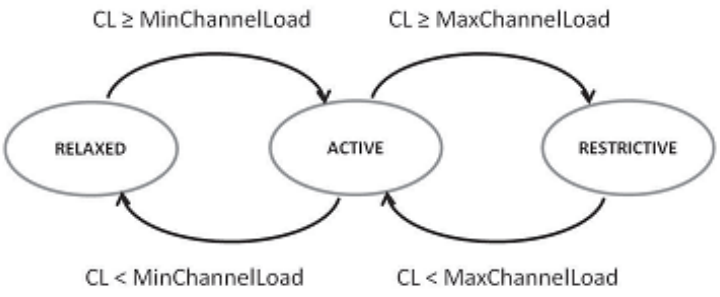

Figure 2.1: The DCC state machine according to ETSI TS 102 687 v1.1.1 [ETS]

2.2 Deterministic MAC Schemes

As shown in the previous section, current standards have been shown to fail in regard to medium access under high channel load conditions. This section explores the MAC schemes that are currently being proposed to solve this problem, as well as what features would a physical device need to provide to implementers in order to be able to create a testbed for the said schemes.

2.2.1 STDMA

STDMA [BUSB09] is a decentralized TDMA-based protocol,based on the Abbreviation Injury Scale (AIS) used in ships, where each node allocates channel time for itself based on its perception of the network and current location and time, as provided by a global navigation satellite system, such as GPS or Galileu. Under this protocol, the medium is divided in time slots, and any station wishing to transmit data must start by assessing the current use condition for each one of these slots. The operation is conducted in four stages: *initialization*, *network entry*, *first frame* and *continuous operation*. The *initialization* consists of listening to all traffic and determine the current slot usage pattern. After that, during *network entry* the station determines which slot it will use to transmit the first packet. This selection tries to use any free slot that may exists inside an adequate time frame, but will fall back to use the slot of the station farther away if no free slot is available, thereby creating deliberate collisions. The next stage, *first frame*, calculates all the remaining slots to use, based on the station transmission rate needs, always attempting to maintain an even time spacing between the selected slots and using a selection algorithm similar to the one in the previous stage. After selecting all the necessary slots for the duration of one frame, the station may enter the *continuous operation* phase. During this phase, each one of the slots previously selected is used up to a random number of times that may range from 3 to 8. After the slot is used the selected amount of times, it is discarded and another slot in its vicinity is selected according to the already described method. This is to account for network topology changes due to stations movement and be able to reconsider the best slot from time to time. If the parameters of transmission are in any way changed, for instance, the messaging rate, the whole process must be restarted from the *initialization* stage. The algorithm used during the *continuous operation* phase can be observed in fig. 2.2.

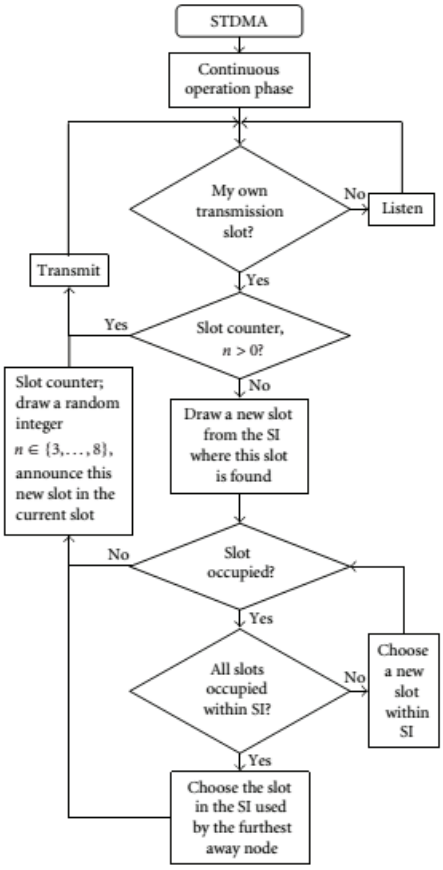

Figure 2.2: STDMA continuous operation phase [BUSB09]

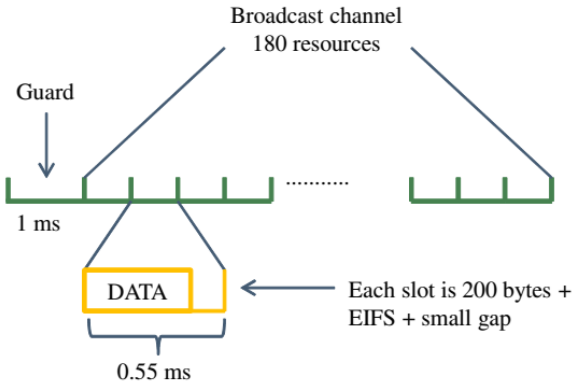

Figure 2.3: Synchronous MAC time domain structure of the channel [SWL+12]

2.2.2 Synchronous MAC

The synchronous MAC [SWL+12] uses an approach of an overlay MAC that provides a slotted structure that allow the transmition of packets in a deterministic synchronous manner without violation the standard MAC mechanism. Under this system, time is divided in frames of 100 ms which are then slotted. Each slot lasts for the intended packet size duration plus the extended interframe space (EIFS), as observed in fig. 2.3.

Each station selects one slot per frame and the packet is only fed to the MAC layer at the slot boundary. By doing this, the standard random backoff mechanism will be avoided and the transmission will start at once due to the following factors:

- A device can transmit a packet without waiting if it senses the channel to have been empty for a duration greater than or equal to EIFS and its backoff counter is zero.
- Since the resource duration is larger than the sum of the packet transmission time and EIFS duration, the channel would be sensed to be idle for at least an EIFS duration at the beginning of any given slot.
- When the MAC gets a packet at the beginning of a slot, its backoff counter would have run down to 0. Note that any post-transmission backoff procedure would have been initiated at least one broadcast interval (100 ms) ago. Furthermore, the slot structure has enough silence periods during a broadcast interval for a device to count its back-off timer down to zero even for the largest possible value. Thus, each device would have a back-off counter that is zero when a new packet arrives and also sense the channel to be idle.

After providing a slotted overlay on the standard MAC mechanism, a resource allocation algorithm is proposed, because it is necessary to decide which transmitters will occupy the same slot simultaneously. While the basic idea is to always use the slot currently used by the farthest station, similarly to the already presented STDMA approach, the following algorithm is used in order to achieve the desired behaviour in a distributed manner [SWL+12]:

- Every device observes the average energy in each resource in the past K broadcast intervals and ranks these slots (resources) in increasing order of observed average energy. Let the total number of slots be N.

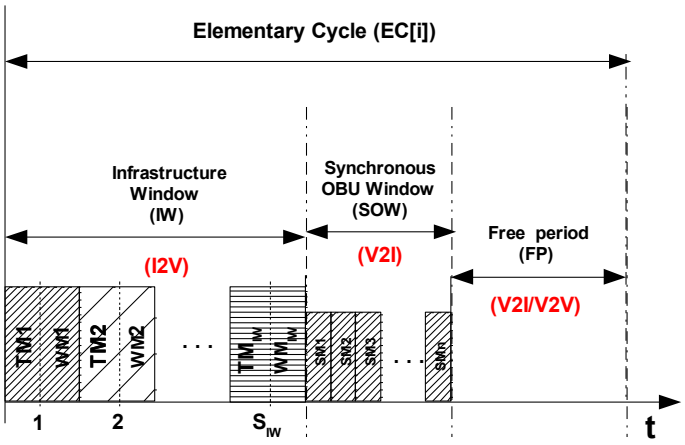

Figure 2.4: Elementary Cycle of the V-FTT protocol [TJJ13]

- Initially, it picks a slot uniformly at random from the first M out of the N slots. Note that these M slots are the ones with the lowest observed energy and typically $M << N$.

- Periodically, a device transmits a slightly smaller packet in its resource so that it can listen to other devices in the remaining time within its resource. If the current slot is observed to not belong to the top M slots due to changes in topology or arrival of other users, it picks a new resource uniformly at random from the top M slots. The broadcast interval in which the smaller packet is transmitted is chosen independently and randomly by the devices so that not all devices transmit the shorter packet simultaneously. The devices transmit one shorter packet in L broadcast intervals on an average.

2.2.3 VFTT

V-FTT [TJJ13] is a multi-master multi-slave Time Division Multiple Access (TDMA) based protocol where the Road Side Units (RSUs) act as masters and schedule all transmissions of the On Board Units (OBUs) which act as salves. Under this protocol, a super frame is initiated by the master stations with a certain periodicity, the Elementary Cycle (EC), and is composed of several phases, as depicted in fig. 2.4.

The first phase of the EC is used to schedule the slave nodes and is used only by the master stations. In order to improve resilience against reception errors, each OBUs is scheduled by several RSUs, which operate in a coordinated fashion to always attribute the same scheduling information to each one of the OBUs. During this window, multiple RSUs transmit scheduling information in sequence. The second phase is used by the OBUs to transmit their status information in the attributed time slot. Contention free behaviour inside these two windows is achieved by not using the regular collision avoidance mechanisms and executing all transmissions in a time-triggered fashion. Resilience against alien stations is also accounted for by always using the short interframe space (SIFS) between adjacent frames. Any station following the standard collision avoidance methods is not capable of compromising the integrity of these cycles.

A third phase, the free period, is used to allow transmission of information by alien

	STDMA	SMAC	VFTT
time triggered packet transmission	x	x	x
receiver sensitivity control			x
global time synchronization	x	x	x
position information	x	x	x
collision avoidance disabling	x		x

Table 2.2: Deterministic MAC requirements table

stations or non safety related information by compliant stations. During this phase the standard collision avoidance mechanisms are used to access the medium.

2.3 Generic deterministic MAC device requirements

Following on the analysing of section 2.2, it is possible to infer about the needed hardware functionalities to implement deterministic MAC protocols.

STDMA requires the ability to bypass the carrier sensing mechanism in order to be able to transmit packets even at the cost of causing deliberate collisions. The distributed time slotted approach requires global time synchronization. The algorithm that manages the whole system is based on location information, which must be available at each one of the stations. At last, being a time division approach, the packets must be sent according to a time stamp value and not in the best effort method devised in the standard approach.

Synchronous MAC is slightly less demanding in the required features and is able to be implemented relying only on accurate global time syschronization and time triggered packet transmission. No bypassing of standard mechanisms is required at all, only a fine control over the instant that the packet is delivered to the MAC layer.

It might also be interesting to provide mechanisms to limit the device effective range in order to reduce the RSU range and thereby the maximum amount of OBUs it must accommodate. This can be done, for instance, by manipulating the minimum signal levels required to perform the carrier sense procedure.

A roundup of all the required features to implement any of the presented deterministic MAC schemes is presented in table 2.2.

Chapter 3

The IT2S Experimental Vehicular Communication Platform

The IT2S vehicular communication platform is a research oriented 802.11p compliant device developed following SDR techniques which uses an FPGA for the baseband signal processing. It is a SoftMAC device in the sense that only the time constrained facilities are implemented in the FPGA and most of the MAC layer is implemented in a general purpose computer connected to the FPGA via a high speed USB connection. Although designed to be standards compliant, the architecture of the platform is very versatile and flexible enough to allow modifications to specific parts of the implementation. In fig. 3.1 an overview of the main components can be seen, with two radio front-ends and a Global Positioning System (GPS) module connected to an FPGA which is itself connected to a general purpose computer via an USB 2.0 connection [FOAC13].

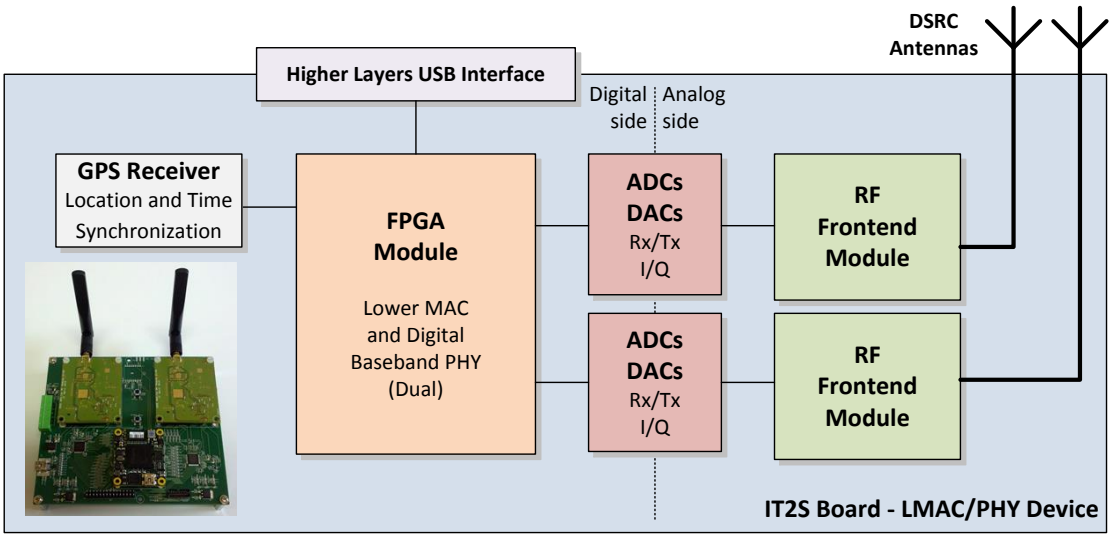

Figure 3.1: IT2S platform architecture

The main goal of this work is to create a device where deterministic MAC mechanisms

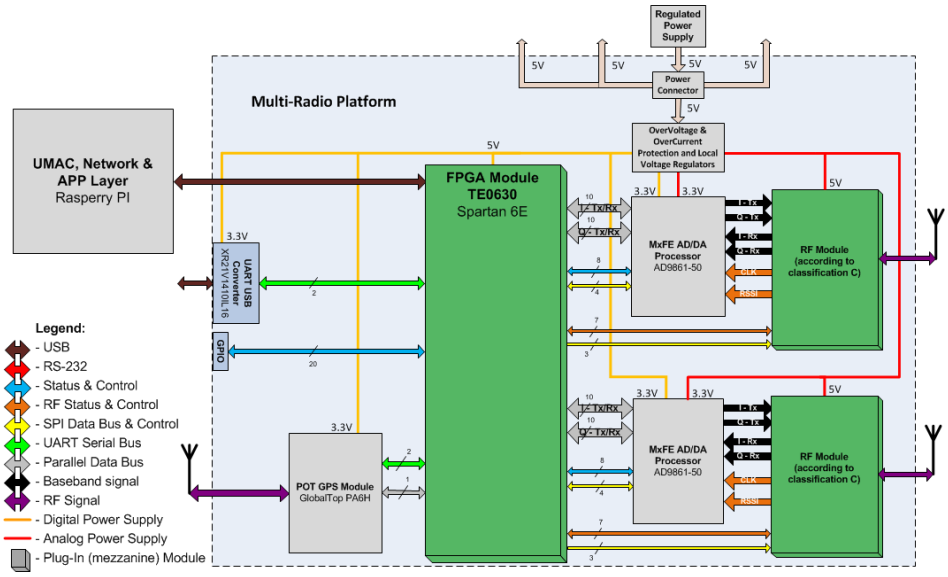

Figure 3.2: IT2S platform detailed architecture [dA13]

---

can be implemented and tested, by using the IT2S platform as a starting point. This chapter will make a presentation of the various platform components and their capabilities and shortcomings, in order to provide a solid basis for the next chapters.

3.1 Hardware components

Although the platform itself is custom design, the components used on its construction are Commercial Off-The-Shelf (COTS) [dA13]. Figure 3.2 shows a detailed architecture of the platform, discriminating the used components.

The centrepiece of the platform is the Trenz TE0630 micromodule. It is an industrial grade device containing, among other things, a Xilinx Spartan 6 XC6SLX150-CSG484, the most capable of the family, 128 MB DDR3 RAM and a Cypress EZ-USB CY7C68013A-56LTXC(I), also known as the FX2LP and non volatile memories to hold both the FPGA bitstream and the USB controller firmware. It provides 56 I/O pins that are used to connect the remaining hardware of the platform to the FPGA using a multitude of protocols, such as Serial Peripheral Interface (SPI) or Universal Asynchronous Receiver/Transmitter (UART).

The main board uses an AD9861 Analog-to-Digital/Digital-to-Analog (AD/DA) to cross between the digital and analogue domains. It provides two 10 bit resolution Analog-to-Digital Converter (ADC) capable of working at up to 200 Msps and two 10 bit Digital-to-Analog Converter (DAC) capable of operating at up to 50 Msps. One ADC/DAC pair is used for the in phase component of the Orthogonal Frequency Division Multiplexing (OFDM) signal while the other is used for the quadrature component. The bandwidth of the channel used in vehicular communications is 10 MHz, therefore, the minimum required sampling rate is 20 Msps, making the ADC more than capable of fulfilling the task. One particular feature of this module is the availability of an auxiliary ADC to process the RSSI signal provided by the radio hardware, that is later used as the main parameter for the MAC mechanism.

This module is connected to the FPGA via an SPI interface, both for configuration and data transfer.

The radio front-end was specifically designed to operate in the frequency band allocated for vehicular communications. It is centred upon a MAX2828 module along with a Power Amplifier (PA) to generate a suitable signal for transmission and an Low Noise Amplifier (LNA) to be able to receive signals within the mandated sensitivity. This module is connected to the AD/DA via a set of differential IOs for optimal performance.

The GPS module is GlobalTop PA6H which although possessing an integrated antenna also supports an external one for improved performance. With this module it is possible to integrate accurate position and time information directly on the platform, with recourse to external solutions. It is connected directly to the FPGA via an UART interface, both for information retrieval and configuration, if needed.

The platform itself also provides a couple more interfaces that can be used to interact with it: a set of General-Purpose Input/Output (GPIO) directly connected to the FPGA which can be used during tests requiring synchronization between multiple devices and an UART which can be used as low throughput data transmission solution between an external computer and the FPGA, either for test or debug purposes.

Finally, being a SoftMAC device, this platform makes use of some sort of computer connected to its USB interface to provide the implementation of the higher stack layers. The device currently used is a Raspberry Pi embedded PC running a GNU/Linux operating system. Despite the fact that the only requirements for a device to fill this role is the ability to be a USB host, this work will focus on using the platform as it is, without any hardware modifications, including to the selected embedded PC. The final solution should, however, be portable to other computers.

3.2 PHY

As explained in section 2.1, the physical layer used in vehicular communications is based on the OFDM PHY specification of the IEEE 802.11 standard. The IT2S platform, using software defined radio centred approach, implements the Physical Layer Convergence Procedure (PLCP) in the FPGA and the Physical Medium Dependent (PMD) layer using both the baseband board and Radio Frequency (RF) frontend.

This work is not concerned with modifications to the PLCP nor to the PMD, therefore, it is sufficient to present the interface that the PHY as a whole provides to the system. The IT2S implementation closely follows the standard description and provides the following standard primitives:

- `PHY_TXSTART`
- `PHY_DATA`
- `PHY_TXEND`
- `PHY_CCA`
- `PHY_RXSTART`
- `PHY_RXEND`

Other than those, it also provides one more primitive, PHY_CHGCHANNEL, with a semantic that is identical to the other ones.

These primitives, clearly defined in the standard [IEE12], are provided via the set of signals shown in fig. 3.3. The INDICATION signals are used by the PHY to inform the LMAC of events that need to be addressed at once, without any other form of handshaking, such as one of the packet reception phases or clear channel assessment. The REQUEST signals are used to start an operation, such as one of the packet transmission phases, be it start, data transfer or end, or channel change. The CONFIRM signals are the PHY acknowledgement for any previous request. The VECTOR signals contain the operation parameters, while the DATA signals are used to transfer the data. Together, this primitives provide access to all the functionality that the PHY has to offer while abstracting the underlying hardware and give plenty of leeway for the MAC implementation. In itself, the PHY does not enforce any specific behaviour, other than the physical signal formatting to ensure interoperability between stations. The PHY_CCA primitive can be used by the MAC layer in order to implement the standard medium access mechanism, but can also be ignored in favour of some external mechanism to assess the medium state. It is completely up to the MAC implementation to decide the best course of action.

3.3 Lower MAC

The current MAC layer implementation used in the IT2S device follows all aspects of the 802.11 standard for vehicular communications [dS11]. Being a SoftMAC implementation, the hardware only contains the time critical components of the MAC layer, or the LMAC, which is responsible for:

- manage PHY interface: transmission, reception, channel changing
- provide memory to store frames
- generate and verify the Frame Check Sequence (FCS) for each frame
- implement CSMA/CA mechanism
- provide suitable interface for the Upper MAC (UMAC)

All memory used to store frames, either to transmit or received, is inside the LMAC. The provided primitives to interact with the memory manager are reduced to slot allocation and release, where each slot is big enough to hold any possible frame. No facilities for message queuing nor separate memory buffers for messages with different priorities are provided. Apart from automatic allocation of memory for received packets, all other memory management functions are performed by the UMAC, namely, memory allocation to store packets to be transmitted and memory releases.

The reception of a packet from the PHY is completely automated and no intervention from the UMAC is required. During reception, the FCS field is automatically checked and any error is signalled to the UMAC. A memory slot is automatically allocated for the incoming packet and later reported to the UMAC to allow the packet retrieval. It is left to the UMAC to later free that slot in order to prevent the memory from being full.

The transmission of a packet starts with the UMAC allocating a memory slot and placing the frame to send in it. It is then followed by a request to send containing the transmission

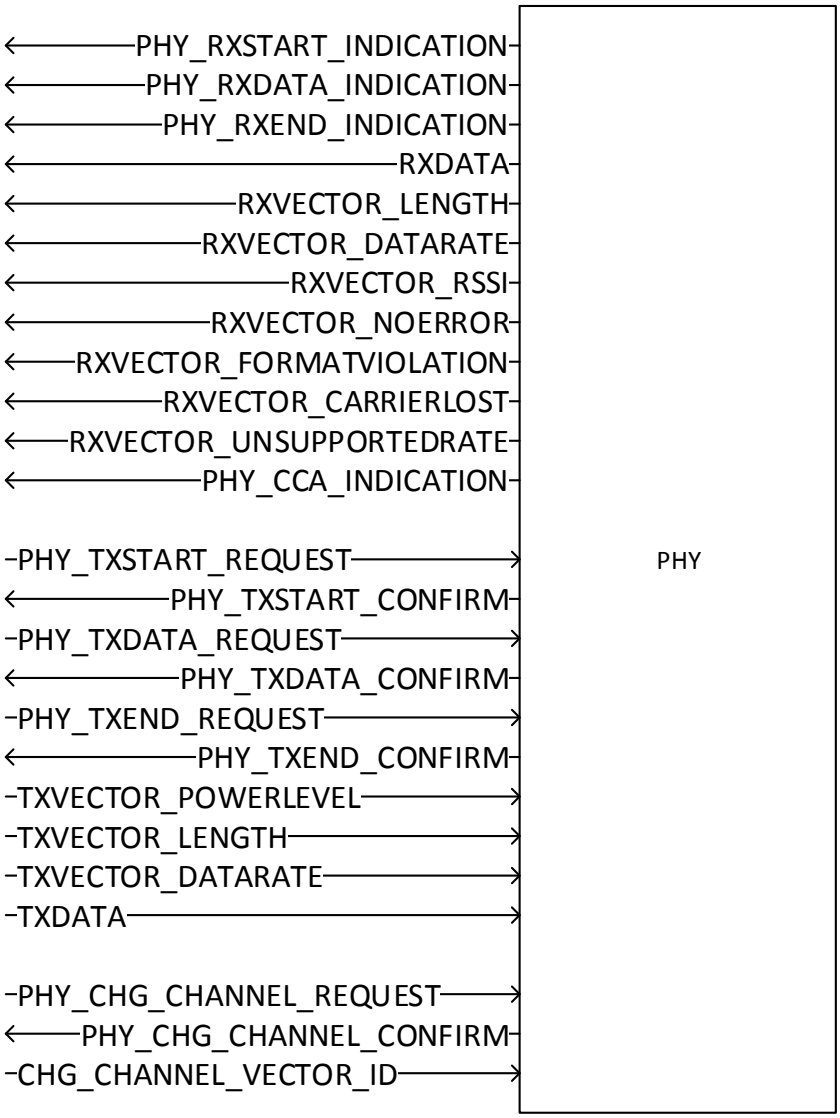

Figure 3.3: PHY interface

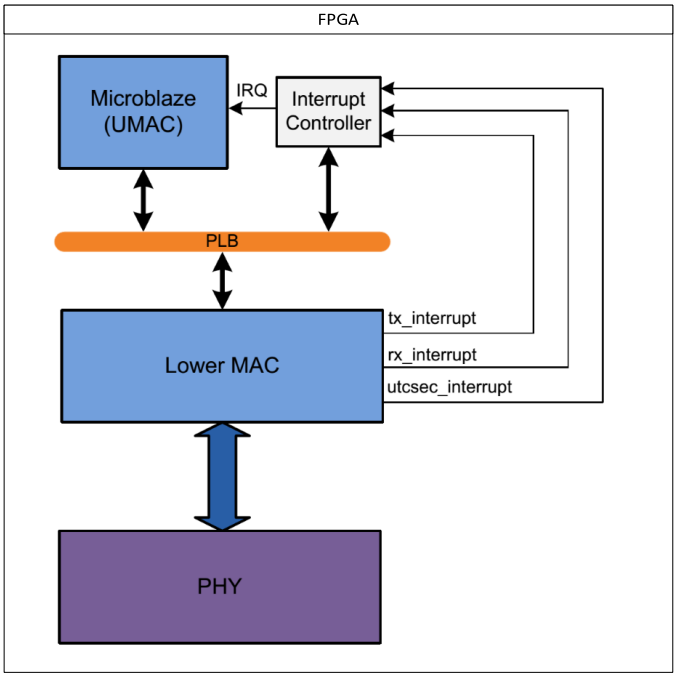

Figure 3.4: LowerMAC interface with other modules [dS11]

parameters. The transmission will occur following standard mechanisms of CSMA/CA and may be postponed indefinitely, until the medium is deemed free for the necessary amount of time. The FCS is automatically calculated and appended to the transmitted frame by the LMAC. After a transmission is completed with success the UMAC is notified and may free the memory slot used to hold the packet.

Channel tuning is also handled by the LMAC. The channel changing operation cancels any ongoing operation, be it transmission or reception and takes the amount of time necessary for the underlying hardware to tune itself in the new frequency.

The LMAC was implemented with the purpose of being integrated in a system where the processor implementing the UMAC lies in the same FPGA fabric as the LMAC, such as the Xilinx MicroBlaze, as shown in fig. 3.4. This decision lead to the design of a register based interface coupled to a Processor Local Bus (PLB) that requires sequential reads and writes to execute each operation. Despite that, the UMAC is currently implemented in an embedded PC that connects to the FPGA via USB, making this interface obsolete and inefficient. It is used simply because of historical reasons.

3.4 USB Connection

The interface between the LMAC and the UMAC is provided by a USB connection between the FPGA and an embedded PC. This link is centred upon a Cypress FX2LP controller running a custom designed firmware, with a matching device driver for the embedded PC and hardware modules for the FPGA.

3.4.1 USB basics

USB is a master slave communication system that is extensively used in both personal and embedded PCs. In a USB system there is only one master, the host and all communications are initiated by it, with a slave node, or *device*, limited to responding the *host* requests. The main reason behind this architecture is to allow all the complexity to remain within the *host*, generally an expensive equipment, while allowing the *devices* to remain simpler and, above all, cheaper. Whenever talking about USB communications, the term *IN* always means a transfer from the *device* to the *host*, while the *OUT* always relates to a transfer from the *host* to the *device*.

The USB wire protocol is packet oriented, providing automatic error checking and retransmission in some cases. The packets are organized in frames which always have the same duration, 1 ms, and are managed by the *host* in order to provide guarantees and allow for the correct functioning of the hole protocol. The high speed version, that is provided by the FX2LP and used in the platform, goes further in dividing each frame in 8 micro-frames, of 125us duration each.

USB distinguishes four transfer types, each one providing a different guarantees and features. **Control** transfers are used by system for configuration purposes and are not used to transfer actual data. **Interrupt** transfers have guaranteed latency, as low as one per micro-frame, but are limited to 1024 bytes. **Isochronous** transfers are lightweight: use a streamlined error checking mechanism and never resend failed packets, but provide guaranteed bandwidth that can be up to 3 1024 byte packets per micro-frame. **Bulk** transfers have automatic error checking and retransmission and use all the bandwidth not used by any of the other transfer types, having the greatest potential bandwidth but also being subject to the current bus usage by other transfer types.

All communications occur over an *endpoint*, a unidirectional link supporting one transfer type. A device using USB to communicate usually provides several endpoints and uses each one of them according to the type of traffic that is transmitting at the moment.

3.4.2 Cypress FX2LP

The IT2S platform uses a Cypress FX2LP device to communicate with the embedded PC. This module implements a versatile high speed USB *device* and provides an adequate interface to interact with the FPGA. At the core of the FX2LP lies a 8051 based microcontroller that not only programs the behaviour of the whole device but is also capable of processing the data coming from the USB connection. In certain applications, this device could be used as standalone processor of a whole system and one can find a wide range of external interfaces to interact with memories or other buses such as Inter-Integrated Circuit (I^2C) or SPI. One other way to use this device is though a special FIFO like interface that was specifically designed to interface complex systems, such as an FPGA. After initial programming, the FIFO interface provides direct access to the 4 general purpose endpoints that the FX2LP is capable of supporting, as shown in fig. 3.5.

Under normal operation, the IT2S platform uses the FIFO interface to allow the FPGA access to the USB and does not use any of the other FX2LP capabilities. There is one circumstance however where the SPI bus is used, and that is when programming the FPGA over USB. That situation is not relevant to this work and will not be covered.

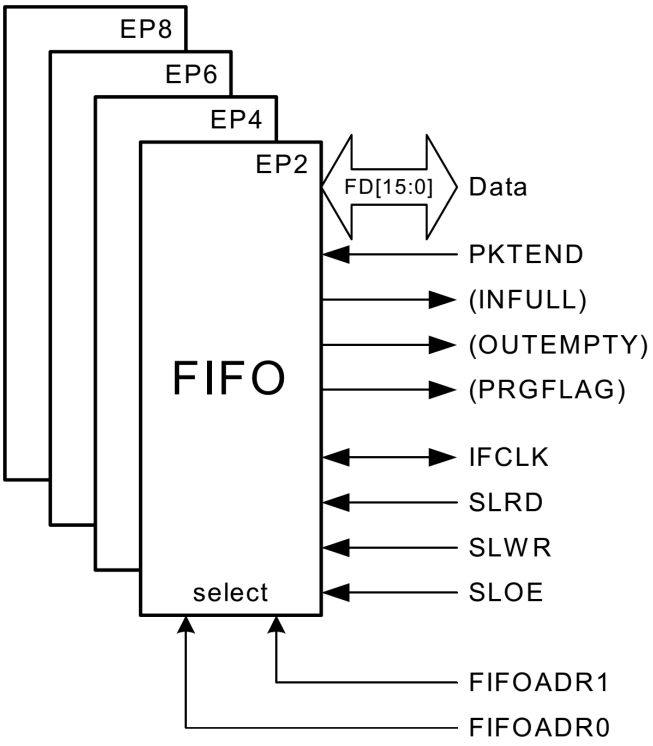

Figure 3.5: FX2LP slave FIFO overview [Sem]

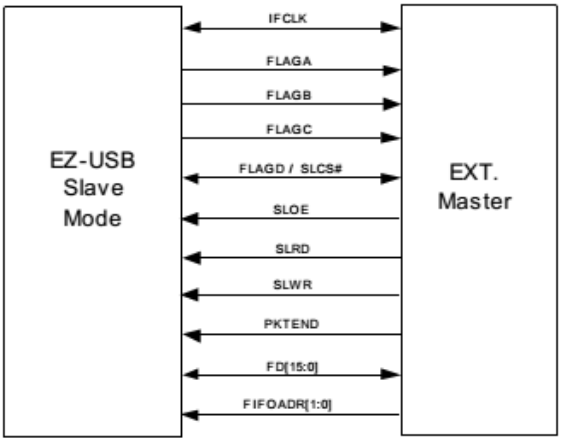

Figure 3.6: FX2 slave fifo interface

3.4.3 Slave FIFO interface

The FX2LP FIFO interface is capable of providing either an 8 bit or 16 bit data bus and some control signals that give direct access to the USB endpoints to an external device, such as an FPGA, as can be seen in fig. 3.6.

The `FD` is a bidirectional data bus that is used to transfer data to and from the FIFOs. Since the IT2S platform hardware only gives access to an 8 bit data bus it is not possible to use the 16 bit solution, only the 8 bit. The direction of the bus is controlled by using the `SLOE` pin. `FIFOADR` is used to select to which FIFO is `FD` currently connected to. The `SLRD` and `SLWR` are use to perform a read and a write operation, respectively. This signals can be configured by the firmware to operate in asynchronous mode and work as strobes, or in synchronous mode and work as enables. Due to the synchronous nature of the FPGA it is advisable to operate the signals as synchronous in order to make timing analysis possible. The FX2LP can be configured to operate either from an interface or external clock source. When driven by an internal clock source, it is possible to make that clock available in the `IFCLK` pin, so it can be used to synchronize the operation of the external master. When driven by an external clock source, that can range from 5MHz to 48MHz, this pin is used to source the intended clock. The flag pins can be configured to provide information relative to the current state of the FIFOs, such as empty and full, or programmable empty and programmable full. The FX2LP will commit data to an *IN* endpoint whenever there is enough to fill an USB packet. It is however possible to force such commit by using the `PKTEND` pin.

All the FX2LP FIFOs have access to a configurable amount of buffering. In the worst case scenario, which occurs when all 4 possible endpoints are being used, this buffering is at most $2x512$ meaning there is space to keep two packets with a maximum size of 512 bytes each. The buffers are packet oriented, so, two packets of 2 bytes will also occupy the complete buffering capacity of this system.

3.5 Device driver

The IT2S platform uses a custom designed communication protocol to allow the LMAC to be controlled by an embedded PC that runs the UMAC. The driver makes use of one *bulk out* endpoint for operations and packets sent by the embedded pc, one *int in* endpoint for events generated by the LMAC, and one *bulk in* to transfer the received packets from the LMAC to the PC. All operations are initiated by the UMAC and the LMAC can only report events related to either the result of the requested operation or a received packet. All information transmitted in the *bulk* endpoints is *stuffed* with the inclusion of start and end of frame markers.

Despite the fact that the platform provides two radios, the current device driver only supports one radio, due to limitations on the communication link between the LMAC and the UMAC.

3.5.1 Communication protocol

The protocol implements 4 operations that directly map to LMAC functionality:

- transmit packet
- receive packet
- change channel
- release memory slot

Each one of these operations is followed by one or more event:

- operation parsing status
- operation deferred
- operation complete

The LMAC can generate on event on its own:

- packet received

The protocol specifically requires that all memory slots are released by the PC, either for transmitted packets or for received packets. This feature was deemed necessary to allow the repeated transmission of the same packet multiple times without requiring it to be resent from the PC to the FPGA, but the facilities to used where never introduced because with time it was considered unnecessary. Currently it simply introduces significant overhead in the protocol.

All operations are executed sequentially in the order they arrive at the FPGA.

3.5.2 Transmit packet

In order to transmit a packet, the PC issues a request containing the intended transmission parameters and the data to transmit. The sequence of message exchange is presented on fig. 3.7.

During this process, when the packet is submitted to the LMAC for transmission, the memory slot it is stored in is reported to the UMAC which is responsible for its later release.

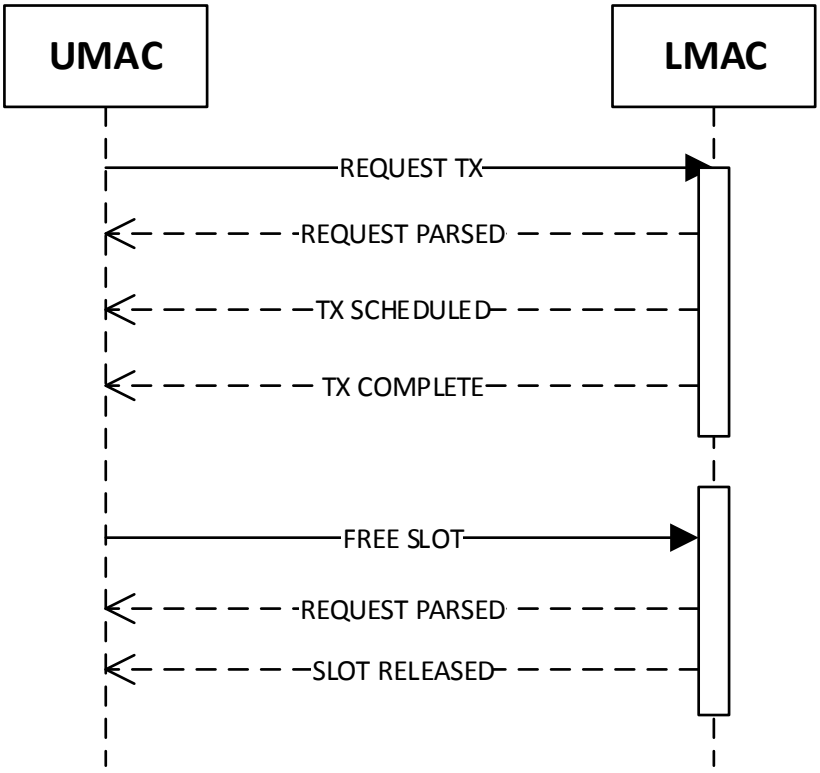

Figure 3.7: Packet transmission operation sequence diagram

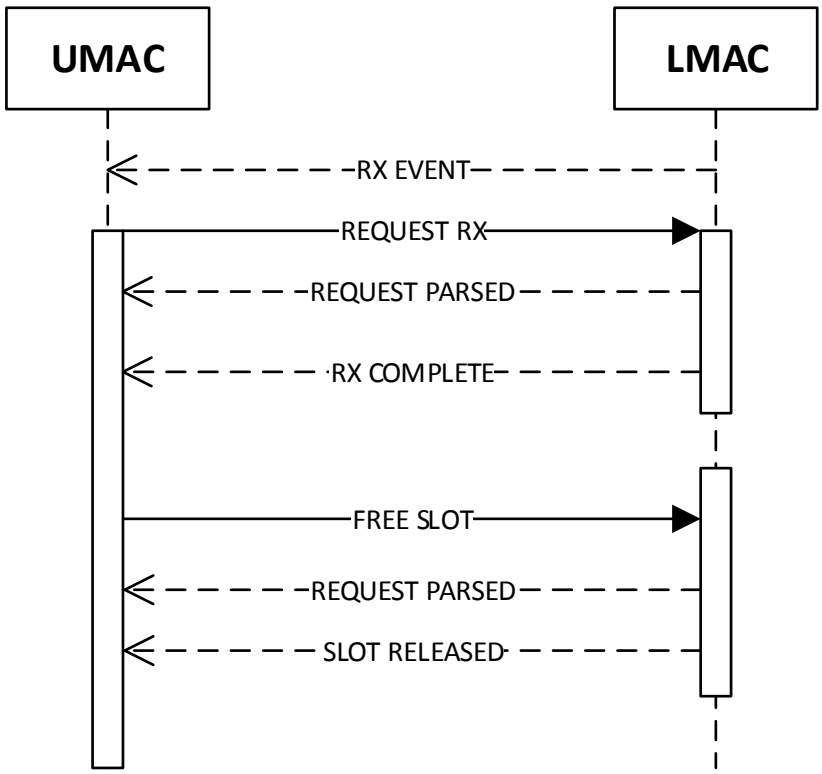

Figure 3.8: Packet reception operation sequence diagram

3.5.3 Packet reception

A packet reception operation starts with an event from the LMAC reporting the received packet status and location. It is then followed by a request from the UMAC to read the said packet and finally with a request to release the memory slot it was using. Figure 3.8 shows the message exchange sequence.

3.5.4 Channel change

The channel change operation is executed by issuing a command with the desired channel. It is then forwarded to the LMAC and any ongoing operation is immediately cancelled. The message exchange sequence can be observed in fig. 3.9.

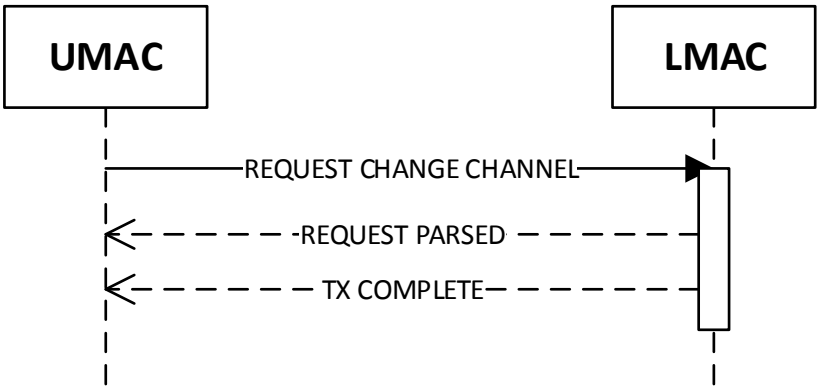

Figure 3.9: Packet reception operation sequence diagram

Chapter 4

Architecture

Having identified both the requirements of a generic deterministic MAC device and the capabilities and characteristics of the IT2S vehicular communications platform, it is now possible to propose on an architecture for the said device, that leverages the flexibility of the existing IT2S platform. The IT2S platform lacks deterministic behaviour due to limitations in both the communication link between the UMAC and the LMAC and the primitives provided by the LMAC. It is necessary to extensively modify the IT2S platform components to be able to provide the functionality identified in section 2.3. This chapter will begin by identifying which modifications are required, and follows to describe them in detail.

4.1 Required modifications to the IT2S platform

The creation of a device that can be used to implement many kinds of deterministic MAC mechanisms based on the IT2S platform requires addressing several challenges, but the modular architecture of the platform allows the separation of the solution in roughly independent modules. The separation used in this analysis is shown in fig. 4.1, with the blocks requiring changes highlighted in grey.

All the studied deterministic MAC mechanisms use the same physical layer, as defined by the standard. This allows the use of the all the platform modules responsible for the generation and reception of information, namely the PHY and all fixed hardware components. On the other hand, the LMAC is tightly coupled to the standard implementation and does not provide the flexibility required to implement a generic MAC mechanism. The existing connection between the LMAC and the UMAC does not provide deterministic behaviour and the device driver does not provide the required facilities to implement a generic deterministic

Figure 4.1: Global architecture overview

MAC. In order to achieve the desired solution, it is necessary to redesign the LMAC and all modules that lie between it and the software implementing the deterministic MAC.

The LMAC, responsible for interfacing the physical layer implementation and the time constrained components of the MAC mechanism must in fact implement all the required functionality and flexibility. As presented in section 2.3, it is mostly desirable to modify the minimum signal level above which the medium is regarded as busy and therefore any packet detected. The device must also be able to disable any collision avoidance mechanisms it possesses in order to be able to transmit packets in a timely fashion and with disregard for any ongoing transmission. At last, there must be support logic attached to a global timing device to allow time triggered transmission of packets. Memories must be provided for several traffic priorities and differentiated between transmission and reception, in order to be able to provide deterministic behaviour for packet transmission.

The link between the LMAC and the UMAC must provide deterministic behaviour and adequate performance to handle the required traffic. The USB communication system currently used by the IT2S platform suffers from several drawbacks that do not allow its use on a deterministic system:

- Use of *bulk* endpoints to transmit all packets to the FPGA fails to provide guarantees on the latency of such transfers;
- The protocol in use is extremely heavy and does not allow the device to be used in its full potential due to lack of bandwidth in the bus to accommodate all the transfers it requires;
- The use of stuffing to identify the beginning and end of transfers introduces non-deterministic behaviour in a sense that the transfer size will vary with the message content;
- Stuffing also introduces a processing overhead, because, while the stuffing/destuffing procedure is extremely simple and efficient to implement in an FPGA, it is slow in a general purpose processor due to its sequential nature. It should be possible to bypass the need for stuffing if some constraints are introduced in the communication;
- The current implementation is only able to support one radio. It is a goal of this new system to be able to support a generic amount of modules, be them radios or other subsystems, such as the GPS module traffic, development tools for debug or any new facilities introduced in future developments.

The device driver must implement the UMAC and provide not only the existing operations but also the primitives identified in 2.3, amounting to the following set:

- best effort packet transmission
- time triggered packet transmission
- modify collision avoidance settings
- change channel
- receive packet

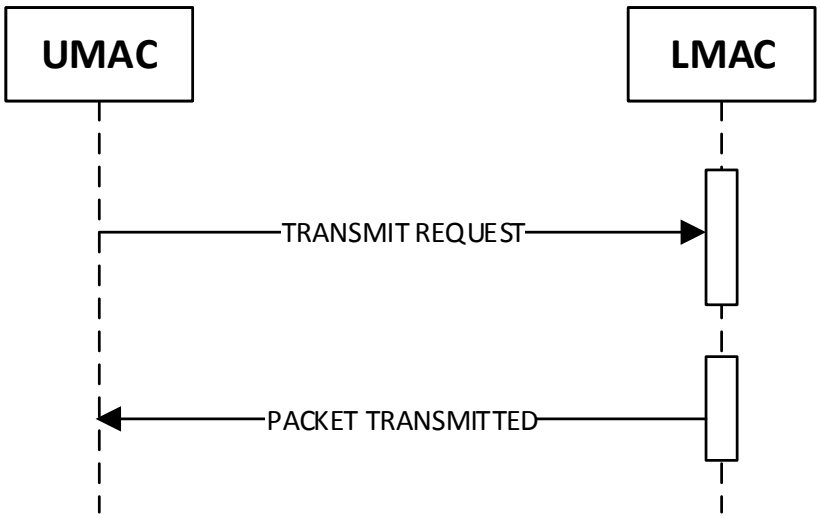

Figure 4.2: Transmission sequence diagram

4.2 Operation protocol

Instead of relying on multiple information exchanges between the PC and the platform for each operation, as the currently implemented protocol, this proposal assumes that both the platform and the communication bus are working properly and issues the commands without waiting for an acknowledgement from the FPGA. It still contains the notion of event which is used for the FPGA to inform the applications about transmission and reception results. As a result of the simplification, only two communication channels are now required, one to transmit data from the UMAC to the LMAC and one in the opposite direction.

4.2.1 Requests

Transmit packet

This request, with a sequence diagram shown in fig. 4.2, is used to transmit a given packet in a best effort fashion. The packet, along with the transmission parameters are sent to the LMAC for inclusion on the transmission queue. Afterwards an event will be generated with the outcome of the transmission. The transmission parameters include: power and bitrate.

Time-triggered transmit packet

This request, with a sequence diagram shown in fig. 4.3, is used to transmit a packet in a specific instant. It is in all aspects equal to the previous request, with the exception that the parameter list includes the time at which the packet should be sent. The device will still

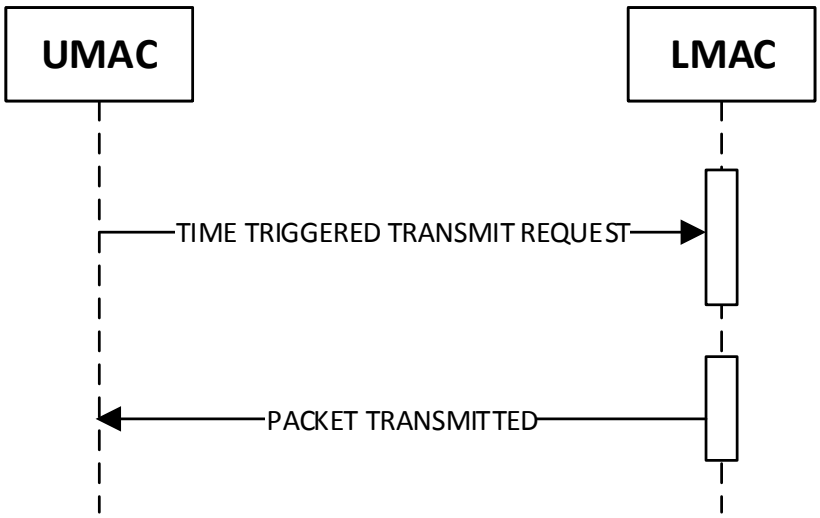

Figure 4.3: Time-triggered transmission sequence diagram

---

honour any collision avoidance mechanism in place, and this request will fail if the medium is declared as busy on the moment of transmission.

Change channel

This request, with a sequence diagram shown in fig. 4.4, is used to tune the device in a different channel. It takes effect immediately upon reception by the LMAC.

Configure CCA

This request, with a sequence diagram shown in fig. 4.5, allows the setting of the device reception sensitivity and carrier sense state. It affects the way in which the LMAC perceives the medium occupancy and after set, these parameters will be used for all transmitted packets.

4.2.2 Events

Packet received

Whenever a packet is received by the platform it is immediately sent to the PC, either just the reception status for a failed reception, or the status plus the data for a successful one, as shown in fig. 4.6. Along with the packet comes also a timestamp containing the instant of the packet reception. This event is in stark contrast with the previously existing protocol, in the sense that the PC automatically receives the packet data without needing to request for

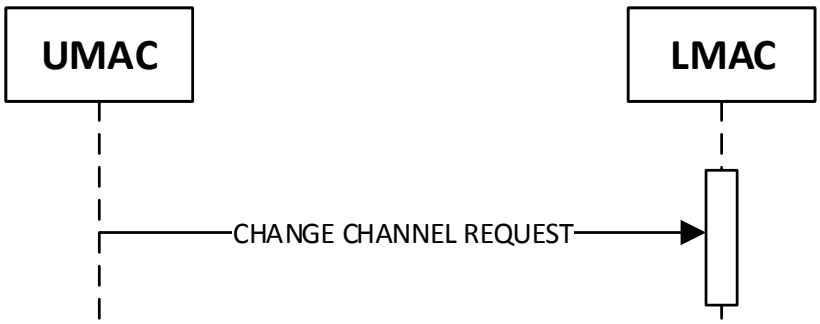

Figure 4.4: Change channel sequence diagram

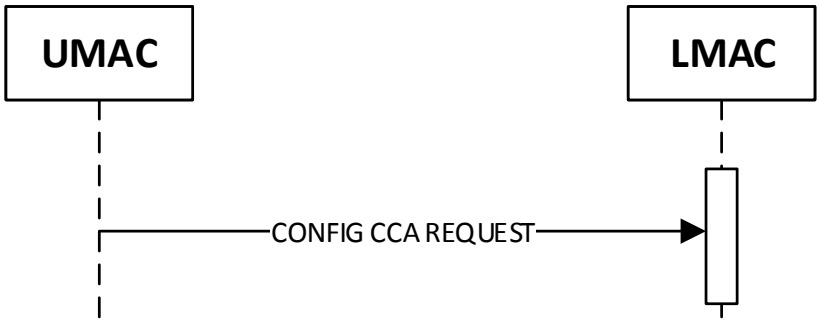

Figure 4.5: Configure CCA sequence diagram

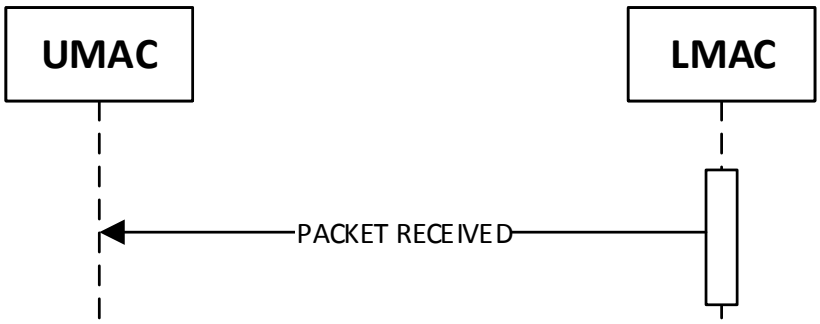

Figure 4.6: Packet reception sequence diagram

---

it. This allows the reception memory to be exclusively managed in the FPGA and reduces the amount of tasks and requests required from the PC.

Packet transmitted

Whenever a packet is transmitted, the transmission status and timestamp and sent to the PC. This allows the application to keep track of the amount of packets still in the transmission queues and also accurate statistics.

4.3 LowerMAC

The LMAC is the responsible for the management of the PHY and for providing all the necessary mechanisms to send and receive data. This section proposes a complete redesign of this module in order to provide the required flexibility to implement generic deterministic MAC schemes. It also includes the architecture of the auxiliary logic required to process UMAC requests and memory distribution. In essence, all blocks inside the FPGA with exception of PHY and the system responsible for providing communication between the LMAC and UMAC, which will be described in 4.4.

The proposed architecture for the LMAC is presented in 4.7. Each one of the modules is responsible for accomplishing a single task and hides the inherent complexity to that task. The separation was made on the basis of allowing future developments without requiring deep architectural modifications. Two modules directly interact with the USB communication system: `Command Processor` and `Event Conveyor`. The first is responsible for decoding and processing of the UMAC commands, while the second conveys the LMAC events and received information to the UMAC. Both modules interface a memory bank, which is were all the required memories are contained. The `Memory Bank` hides underlying complexities of memory access under a convenient interface while at the same time providing simple memory management facilities, and is used to store not only the packets for transmission but also

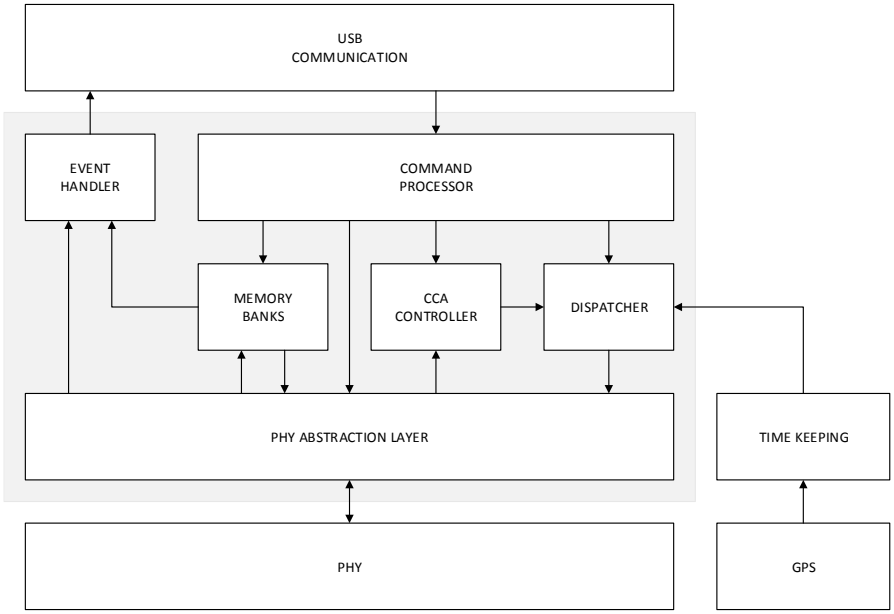

Figure 4.7: Lower MAC and auxiliary logic overview

---

the received ones. The `CCA Controller` provides a flexible assessment of the current channel state allowing enabling and disabling of carrier sense mechanism as well as setting of the reception sensitivity. The `Dispatcher` is responsible for implementing the various priority queues and issue the packet transmission requests, either at a specific time for time triggered transmissions or in a best effort basis for regular transmissions. The operation of this module is regulated not only by the CCA but also by an external time keeping device that provides the time base for time triggered packet transmissions (the design of the time keeping module is out of the scope of this document). At last, the `PHY Abstraction Layer` is responsible for abstracting the PHY interface and providing an adequate set of signals to be used by all the other modules.

In order to implement a `Memory Bank` that suits the needs of the device, it is necessary to make an assessment about the required memory. The communication system is designed in such a way that real-time traffic can be transferred to the UMAC faster than it is received by the LMAC. As such, even considering 8 packet real-time queues in each direction, with a maximum packet size of 512 bytes, the amount of required memory is 8KB. Non real-time packets can be, at most, 2344 bytes long, according to the standard, therefore, making a naive analysis, in order to provide space for 8 of them, at least 18752 bytes are required. So, at the very least, with two non real-time packet queues, one for transmission and other for reception, there is the need for about 36KB of memory. To provide the full set of 8 transmission priority queues and one reception queue under the same circumstances, about 1.3MB of memory are required.

Endpoint number	Type	Direction
2	OUT	Interrupt
4	OUT	Bulk
6	IN	Interrupt
8	IN	Bulk

Table 4.1: Available USB endpoint configuration

4.4 USB connection (Multilink)

Although this work is focused on the development of a device capable of implementing a deterministic MAC scheme, the IT2S platform contains two radios that must be used simultaneously in order to comply with the current standards. The platform also possesses internal testing tools, that although outside the scope of this document, require the transfer of large amounts of information to and from the PC. At last, being a research oriented platform, it may incorporate more modules, both inside the FPGA fabric or using some of the available connectors. It is desirable to develop a communication system between the FPGA and the PC that is capable of accommodating a generic amount of devices with different requirements. This section presents an USB communication system that is capable of transferring both bounded latency and high throughput traffic to multiple independent users over a single USB connection, thus providing an appropriate communication system not only for a deterministic MAC implementation but also for other high throughput users, from now on named `MultiLink`.

4.4.1 Overview

As explained in section 3.4, the USB specification already provides independent communication channels over a single physical link, the endpoints. Each one of this *channels* is capable of transmitting information in one direction and with a certain amount of guarantees, depending on its own type. The IT2S platform uses an FX2LP USB controller that is capable of supporting up to 4 endpoints in the FIFO interface that is available to the FPGA. The proposed solution, shown in fig. 4.8, provides a configurable number of *links*, each one with of two types of *bidirectional* channels: high throughput and bounded latency. At first sight, these channels map directly into endpoints: *bulk* for the high throughput and *interrupt* for the bounded latency, but if one needs to provide more than one *link*, the FX2LP no longer provides the necessary amount of endpoints. Changing this component is not an option, so, to be able to provide more than one link, in this case a generic amount of links to allow for future work, it becomes necessary to devise a scheme that is able to multiplex the native endpoints in some way to achieve the necessary amount of channels.

The number of channels required for each link matches the available endpoint count, and it is possible to individually configure each endpoint to provide the required characteristics for each channel. The approach that this architecture proposes is to configure each one of the available endpoints as presented in table 4.1 and multiplex them to provide the amount of channels required by the whole system.

The architecture of this subsystem, presented in fig. 4.9, is made of three main components that will be separately presented: the FPGA module, the FX2LP firmware and the device

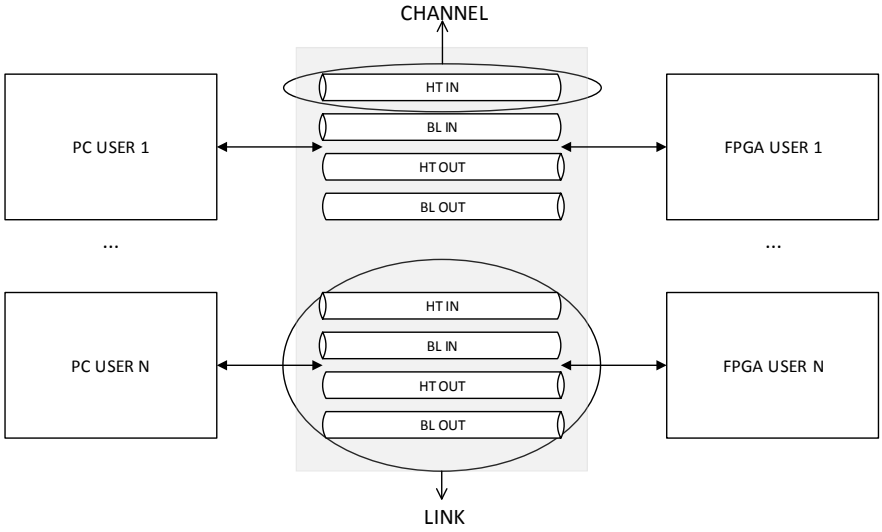

Figure 4.8: `MultiLink` concept

driver. On the FPGA side, a set of FIFOs allow access to each one of the channels, while on the Personal Computer (PC) side, a small set of function calls is provided. The system must provide adequate internal buffering to conceal the inherent latency of the underlying USB connection from the application, both on the FPGA side and on the PC side. The firmware running on the FX2LP is not in any way visible to the user and all explanations concerning it will be left to the next chapter.

4.4.2 Multiplexing protocol

In order to multiplex each one of the USB endpoints, it is necessary to devise a protocol to wrap all the transmitted data and allow the identification of the link to which the said data belongs to.

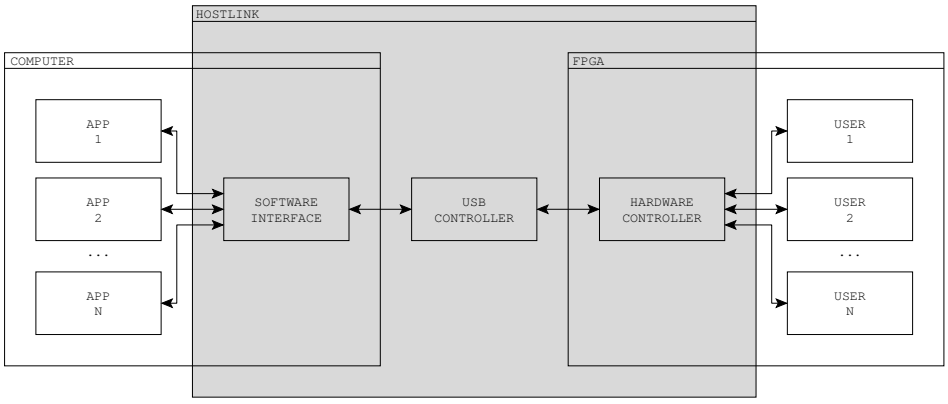

Figure 4.9: `MultiLink` architecture overview

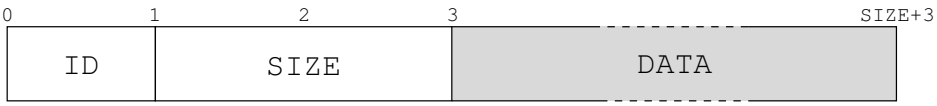

Figure 4.10: MultiLink frame format

The underlying USB protocol is packet oriented and will fragment any transfer bigger than the maximum transfer size which is 1024 bytes both for interrupt and bulk transfers. This fragmentation is done automatically by the USB controllers but is exposed to the users when reading data from the endpoints. On the software side, a single read might not return the full transfer but only the last received USB packet, so the size of the read cannot be used to infer the total amount of transmitted data. On the FPGA side data is always read byte by byte, as provided by the slave FIFO interface.

Bounded latency communications over USB require the use of interrupt endpoints, as already explained. However, it is important to keep in mind that USB only provides guarantees for a single transfer, that is, if the message needs to be fragmented over multiple transfers it is not possible to ensure deterministic behaviour because the latency will be related to the amount of transfers required to complete the operation. Only by keeping the message size small enough to fit in a single transfer is it possible to provide a maximum latency guarantee per message.

High throughput transfers also benefit from this fragmenting behaviour by allowing the dimensioning of the buffers required to keep the data transmitted over the bus and the balancing of the load between different users elegantly as long as the bus remains capable of sinking messages faster than they are produced. This would not happen if messages were of unlimited size and a single user would be able to starve all other users even if producing data at a low rate.

The devised protocol frames all information in messages that are, at most, as big as the maximum USB transfer size, so that one single frame is sent in one transfer and never fragmented. For a packet to be sent, one of two things must happen: either there is enough information to fill an USB packet or the user specifically requests to send a message which size is less than the maximum transfer size. The data is then framed and a header with two pieces of information is added: message size and link identifier, as shown in fig. 4.10. In effect, the maximum payload size is the USB transfer size minus the header size. The size of this header is dependable on the implementation requirements for maximum amount of links.

One of the drawbacks of this solution is that if a packet oriented protocol is used over this system, its messages may be fragmented. It is, however, a similar behaviour to most transport protocols [citation] and any application using it must take appropriate measures that allow the reception of a single packet of information in a fragmented way.

4.4.3 FPGA logic

As explained in section 3.4, the FX2LP provides a slave FIFO interface that can be used to interface an FPGA. It is the task of the FPGA logic of this module to translate that interface into one more suited to be used by other modules in the FPGA. One very common and versatile interface is the FIFO. It provides buffering and allows crossing domains with not only different processing speed but also different clock signals, if necessary. The FPGA logic of this module is then responsible of implementing the multiplexing protocol described

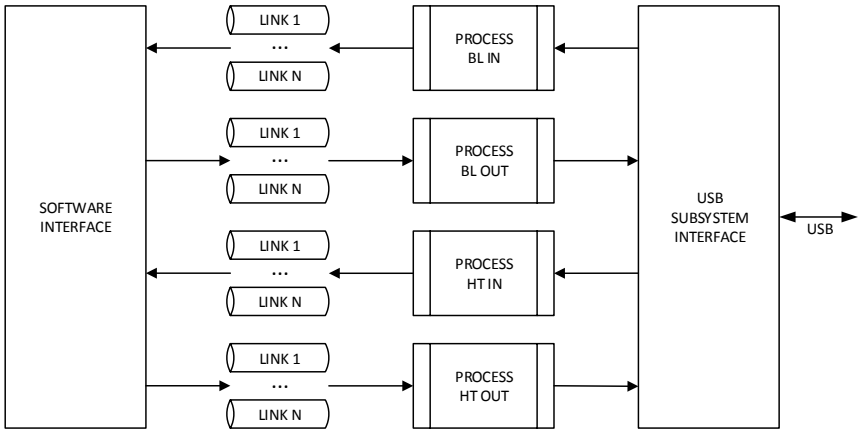

Figure 4.11: `MultiLink` driver architecture

in the previous section and provide adequate buffering for all channels of all links.

4.4.4 Software interface

The software interface to this USB based communication system, or *device driver* provides an interface to communicate with the FPGA with the already explained features. This piece of software should also conceal the underlying USB details and implement the protocol explained previously. The proposed architecture for such driver is shown in fig. 4.11.

The interface provided to the system users allows reading and writing to a set of intermediate buffers. Each one of these sets is managed by a different process that creates appropriate frames in the case of *out* channels or processes incoming frames in the case of *in* channels. As with other transport protocols, a write of a size greater than the underlying maximum transfer size is automatically fragmented into multiple frames. Each one of these processes interacts with one single process that controls the USB link.

4.5 Device Driver

The device driver is responsible for exposing the platform functionality to the user applications using the already described USB communication system. The proposed protocol requires two channels per radio, one to send requests and one to receive events. The type of channels to use is related to the purpose of each one of the radios. Following the ETSI ITS G5 specification, one of the radios is always synchronized on the control channel while the other is used in the service channels. Deterministic MAC schemes will most likely be used only on the control channel, therefore it makes sense to use bounded latency channels for one of the radios and high throughput for the other. The proposed driver architecture should be adequate for both deterministic and high throughput operation.

The driver, which architecture is shown in fig. 4.12, is composed of several separate processes that handle each one of the supported operations. The transmission of the requests to the LMAC is done under supervision of a `Request Scheduler` in order to allow fair access to the used `MultiLink` communication channel. The events sent by the LMAC are separated

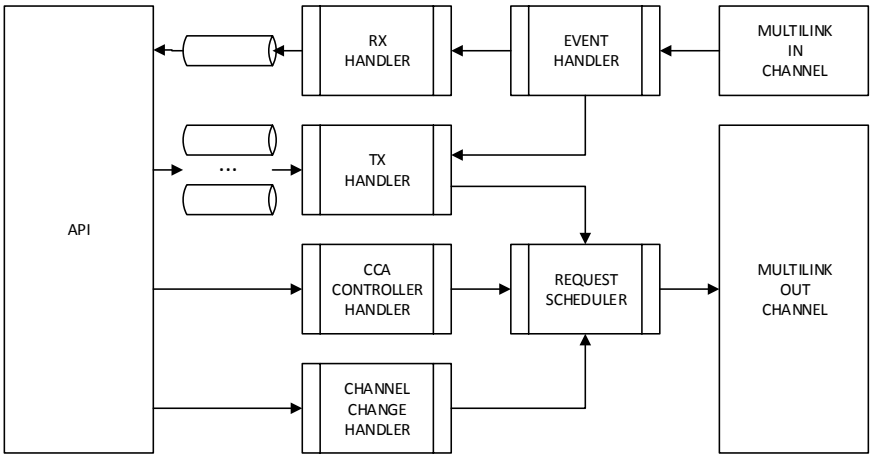

Figure 4.12: Device driver architecture

according to their type and forwarded to the appropriate handler. Transmission events are forwarded to the `TX Handler` in order to allow the maintenance of accurate information regarding the memory state of the LMAC packet buffers while reception events are forwarded to the `RX Handler` in order to update the reception statistics and provide the add the received information to the reception queue..

All this infrastructure is concealed under an interface that ultimately provides the application model to implement the deterministic MAC scheme, as is the main objective of this work. The Application Programming Interface (API) that this system provides is comprised of the following primitives:

- `device_list`: list devices connected to computer
- `read_rx_statistics`: read the packet counters of a device
- `read_current_channel`: read the currently tuned channel of a device
- `rx`: blocking packet read
- `tx`: transmit best-effort packet
- `tx_tt`: transmit time-triggered packet
- `change_channel`: tune device in a new channel
- `config_cca_controller`: configure CCA controller parameters

In can be noted that there are not methods for opening or closing a device. This follows from the use of the `MultiLink` communication system that abstracts the platform from the underlying USB device, therefore, it is possible to have multiple radios using the same physical device. To use this interface, the applications must first call `device_list` to get handlers for the currently connected devices and use those same handlers later when calling any of the other primitives. From the point of view of this interface, multiple simultaneous users of a

single device are allowed and it is left to the higher layers the management of such behaviour in order to ensure coherent operation.

Chapter 5

Implementation

As outlined in the previous chapter, the proposed architecture is divided in three main components:

- Lower MAC
- USB communication system
- Device driver

This chapter will explain in detail the implementation of each one of these elements.

5.1 Lower MAC

The implementation of the LMAC follows the description presented in section 4.3.

After having presented the architecture of the LMAC in section 4.3 it is left to the implementation to define the inner structure of each one of the described blocks and their interconnections. All the blocks presented here are implemented in the FPGA and as such have no restrictions concerning which internal interfaces are used. Each one of the developed blocks provides a custom interface that better exposes its functionality to the other blocks and does not suffer from constraints introduced by the use of standard interconnects. Most of the blocks are implemented using a state machine to achieve the desired behaviour.

5.1.1 Memory Bank

The FPGA module under use provides two types of memories that meet the requirements outlined in 4.3, namely: external Double Data Rate (DDR) and internal Block RAM. The DDR provides much more capacity than the block RAM, 128MB vs 523KB, but must be accessed via a specific interface that allows only one access at a time. In order to use this memory, it would be necessary to use specific blocks and implement a multiplexing mechanism that would allow simultaneous access to both the transmission and reception paths. The Block RAM on the other hand, exposes itself as configurable amount of independently sized memories that can be tailored to match the user requirements. However, this memory is also used by other FPGA blocks, such as First In First Outs (FIFOs) and not all of it is available to be used by the memory bank. According to the already referred space requirement analysis, 44KB (36KB + 8KB) should be enough for a basic working implementation, with

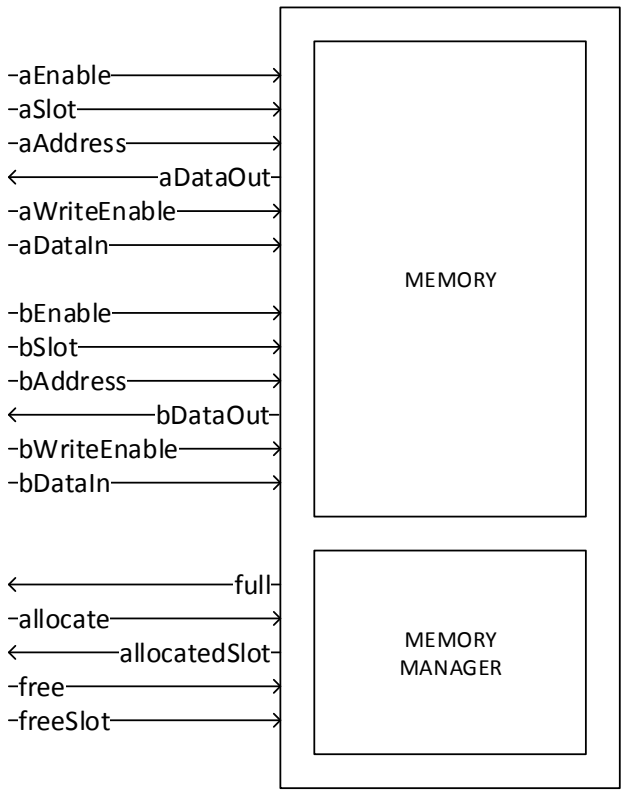

Figure 5.1: Memory bank block diagram and interface

only one transmission priority, amounting for roughly 14% of the available Block RAM. In order to keep the implementation simple this memory is going to be used to provide the packet buffers required for the device. The modular design of this system allows for future expansion according to the perceived needs.

The interface provided by this module, shown in fig. 5.1, provides two independent ports per memory in order to allow simultaneous access to both the `PHY Abstraction Layer` and the `Command Processor` or `Event Handler`, depending on the memory type. There is one instance of this module for each required packet type and priority, therefore providing independent space for each one of the packet types. The system is organized in slots. Each slot is big enough to hold a packet of the maximum allowed dimension and has an unique ID that allows the calculation of its initial address. The memory is byte addressable and in each cycle one byte is read or written. Along with the memory, this block also provides the memory management facilities required to maintain track of the current memory usage. Allocation and release is made one slot at a time and signals are provided to inform of full memory condition. The timing diagram of an allocation is present in fig. 5.2.

The block diagram of the memory manager implementation can be observed in 5.3. Contrarily to other blocks in the LMAC this module was implemented with basic logic blocks due to its simplicity. The proposed implementation achieves allocation and release of memory slots in a single clock cycle.

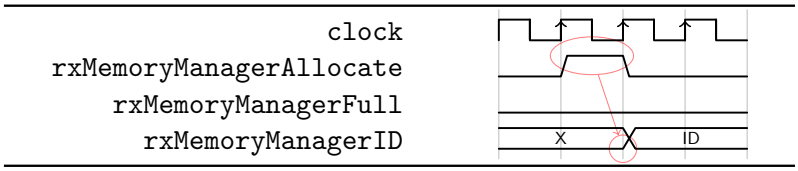

Figure 5.2: Slot allocation timing diagram

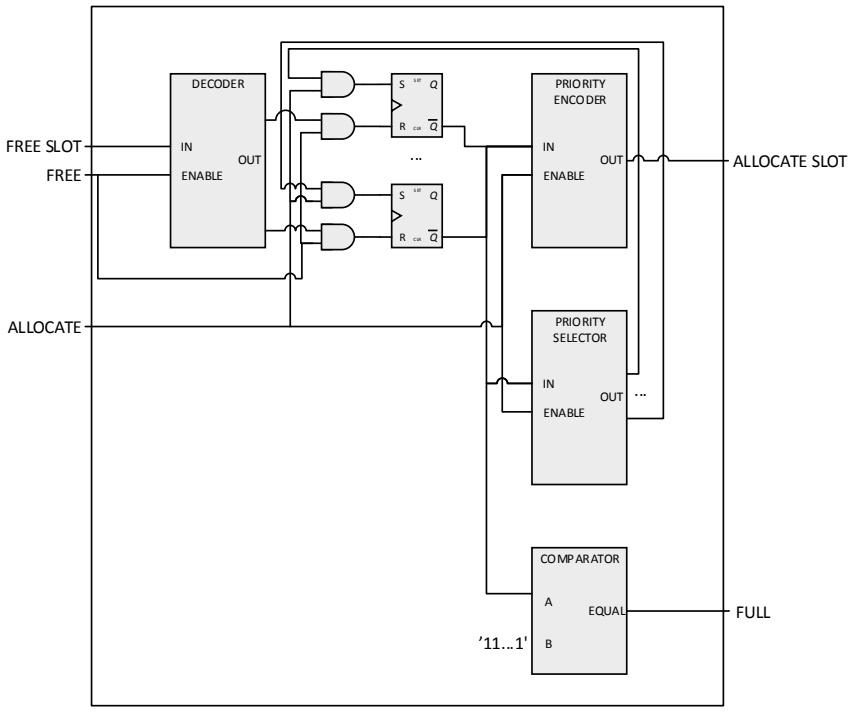

Figure 5.3: Memory manager block diagram

5.1.2 Command Processor

The processing of the UMAC issued commands is made using a state machine presented in fig. 5.4, that parses each one of the commands parameters and drives the other blocks of the LMAC. The commands are executed in the order in which they are received and it is assumed that the protocol is followed without errors. No blocking ever occurs inside this state machine other than waiting for data from the UMAC whenever necessary, therefore, any command that can't be immediately executed is silently ignored.

While modifications to collision avoidance mechanism and channel changes can be executed at once, packet transmission requests must be scheduled. In this case the packets are stored in the memory and the transmission parameters are added to the respective queue for the dispatcher to handle the transmission itself.

5.1.3 Dispatcher

After each packet to be transmitted is stored in the appropriate memory, its identifier, together with the respective transmission parameters, are added to a queue that corresponds to the transmission priority. The queues are then emptied following a fixed priority mechanism that is implemented by a scheduler. The scheduler is bound to a global time reference to perform the time-triggered transmissions. After a packet is transmitted, either successfully or not, the memory slot it occupied is automatically released. The current implementation of this module only supports two priority levels, either best effort or time triggered. The architecture is however in place to be able to support a generic amount of priorities in future developments. As observed in 5.5, there is one set of signals for each one of the queues that store the packet information, also one set to access the respective memories one set to control the PHY and at last, the real time clock (RTC) signal to trigger the time triggered transmissions.

5.1.4 CCA Controller

This block monitors the RSSI and provides an flexible CCA signal whose behaviour can be configured by the UMAC. The configurable parameters are the current threshold and the carrier sense state. Its implementation consists of one register containing the current RSSI threshold coupled to a comparator. The medium is considered busy whenever either the reception chain senses the carrier and carrier sense is enabled or when the RSSI level exceeds the configured threshold. The device range can then be limited by disabling carrier sense mechanism and programming an appropriate threshold on this module.

The reception chain is constantly decoding the received information, therefore, in order to achieve the desired range limitation effect, when the carrier sense mechanism is disabled, it is also necessary to disconnect it from the information received from the ADC whenever the RSSI is below the predefined threshold. This effectively inhibits the decoding of any received information without modifying the reception chain itself. By definition, if the carrier sense mechanism is enabled, the range is only limited by the sensitivity of the reception chain.

5.1.5 Event Handler

Each packet transmission and reception generates an event that must be conveyed to the UMAC. Packet transmission events serve the purpose of maintaining accurate memory

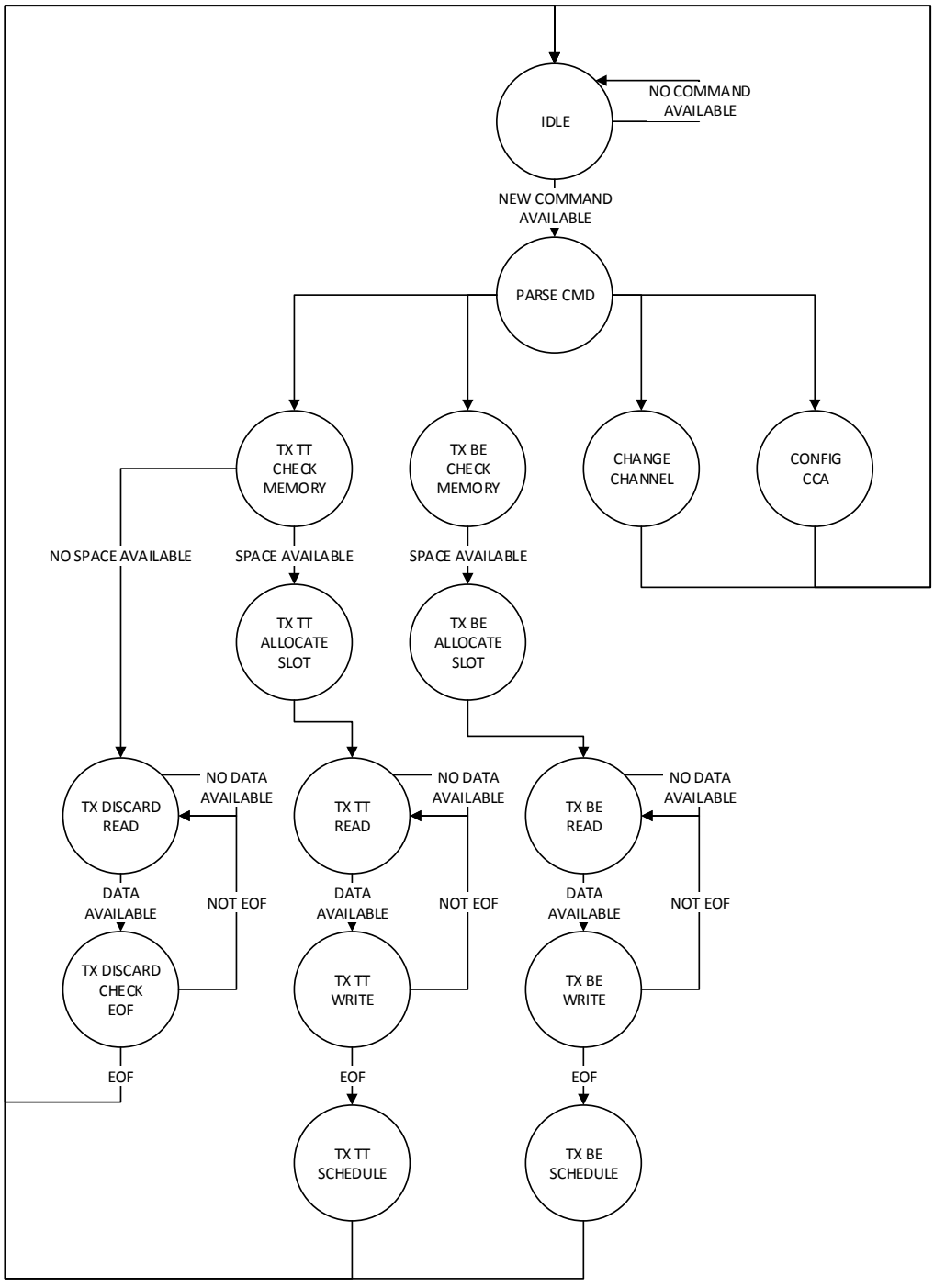

Figure 5.4: LMAC command processor state machine

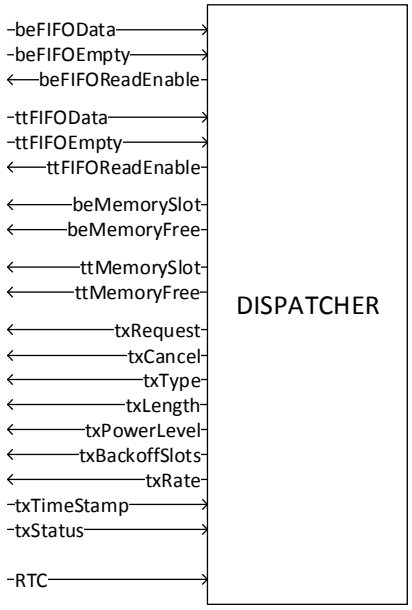

Figure 5.5: Dispatcher interface

occupancy values in the UMAC while packet reception events are used to send the received packets to the UMAC. On top of event conveyance, this block also releases the memory slots occupied by the received packets after sending them to the UMAC.

5.1.6 PHY Abstraction Layer

This block is mainly a shell around the real PHY interface with a few additional features that allow automated access to the already mentioned packet memories, calculation and verification of the FCS and that also provides the collision avoidance mechanism as needed.

As observed in fig. 5.6, the interface is divided into several groups of signals, plus a real time clock signal. There is one group for each one of the supported operations and events, two groups that allow access to each one of the memories and another group containing the rx memory manager signals. At last, there is one signal that contains the current real time clock value. Each one of the supported operations uses a `request` signal that validates all the other input signals of the group and initiates the respective action. On those groups, the `status` signal is updated as the operation progresses. Ultimately, the status signal marks the end of the operation with either success of failure. The events generated by this block use a `ready` signal is that validates all the other output signals of the group, including the status signal. The memory manager interface was already described and should be straightforward.

Packet transmission

In order to transmit a packet, the identifier of the memory slot in which it is stored along with the parameters to be used for transmission must be provided. The transmission may be cancelled at any time using the `txCancel` signal. The current status of the transmission

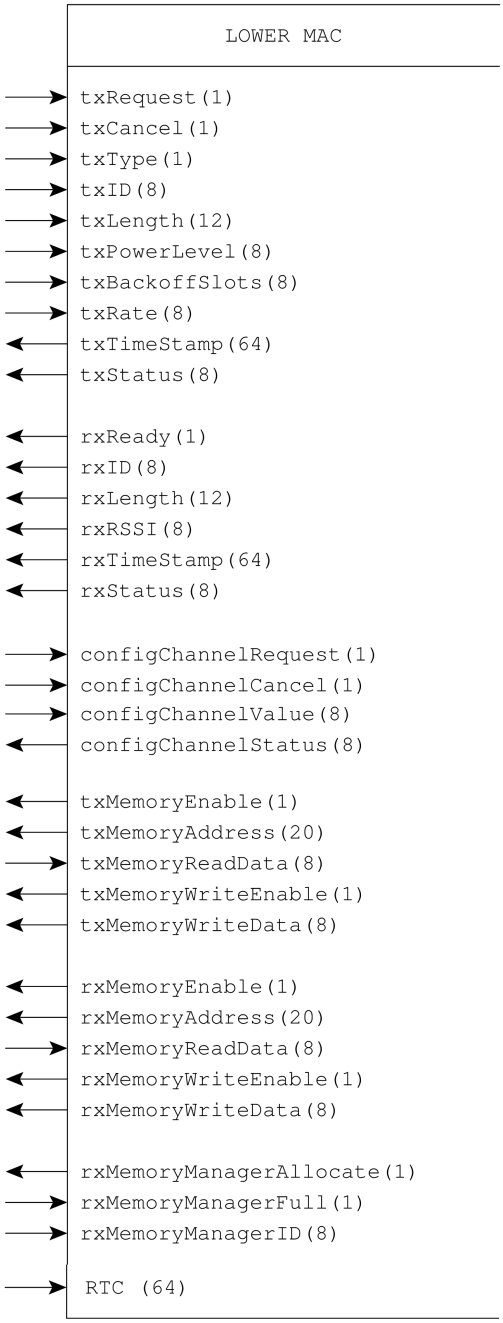

Figure 5.6: The module interface

is reported via de `txStatus` signal. What follows is a detailed description of each one of the interface signals.

- `txRequest` initiates a new transmission. This request is ignored if a previously requested transmission has not yet completed with either successful, failed or cancelled condition.
- `txCancel` is used to cancel an ongoing transmission. The transmission may be cancelled at any moment, but the time it takes to do it varies with the current internal state of the LowerMAC. `txStatus` should be checked to detect the end of the cancellation.
- `txType` is 0 for non real-time and 1 for real-time transmissions. When a real-time transmission is requested, the collision avoidance mechanism is not executed and the transmission fails immediately when the medium is occupied. On the other hand, when the medium is free, all internal wait sates mandated by the standard are bypassed and the frame is transmitted immediately.
- `txID` is the memory slot where the packet to be transmitted is stored.
- `txLength` is size of the packet to be sent, in octets, not including the FCS.
- `txPowerLevel` is the power level to use. The correspondence between this value and real power is present in fig. 5.7.
- `txBackoffSlots` is the upper bound for backoff duration in slots. Each slot corresponds to 13μs, as specified in section 18.4.4 of the IEEE 802.11-2012 standard [IEE12].
- `txRate` is the signalling rate to use according to table 5.1.
- `txTimeStamp` holds the exact moment at which the first energy burst was sent over the air, accoding to `RTC`. It is only valid when the `txStatus` signals that a successful transmission just ended.
- `txStatus` is a bit field, shown in fig. 5.8, that informs the user about the internal state of the LowerMAC and to the outcome of the operation. Only one bit of `txStatus` is asserted at a time. `CONTE` is asserted during the contention phase. `ONGOI` is asserted during the physical transmission. `SUCCS` is asserted at the end of a successful transmission, during one clock cycle. `FAILE` is asserted at the end of a failed transmission during one clock cycle. `CANCE` is asserted at the end of a cancelled transmission during one clock cycle.

Figure 5.9 depicts a possible timing diagram for a successful non real-time transmission. As the shown, after requesting a transmission, the device contends for medium access waiting for its opportunity to transmit. When the medium is clear, the transmission is carried out. At the end, `txStatus` signals a successful transmission and the time stamp is reported.

A failed real-time transmission timing diagram can be observed in fig. 5.10. In this situation, right after the transmission request, `txStatus` changes to signal a failed transmission.

At last, a cancelled transmission timing diagram is present in fig. 5.11. This one is similar to the successful transmission, except for the fact that `txCancel` was asserted at the middle of an ongoing transmission and shortly after `txStatus` changed to inform about a cancelled transmission. No time stamp is provided in this situation.

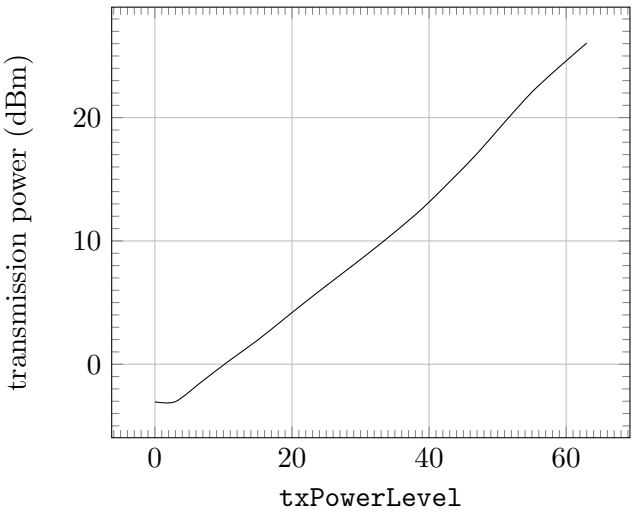

Figure 5.7: `txPowerLevel` conversion to real power

`txRate`	Modulation	Rate (Mbit/s)
0	64 QAM 2/3	24.0
1	16 QAM 1/2	12.0
2	QPSK 1/2	6.0
3	BPSK 1/2	3.0
4	64QAM 3/4	27.0
5	16 QAM 3/4	18.0
6	QPSK 3/4	9.0
7	BPSK 3/4	4.5

Table 5.1: Signalling rate information

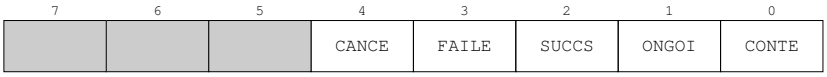

Figure 5.8: `txStatus` fields

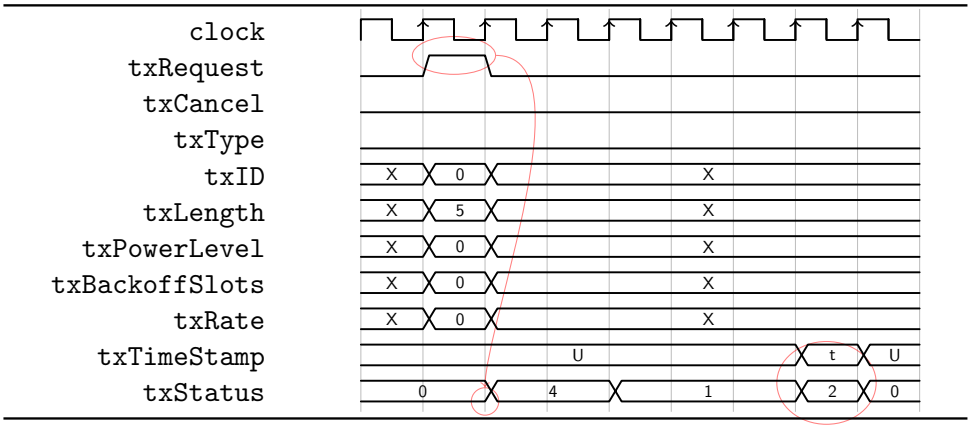

Figure 5.9: Successful non real-time transmission timing diagram

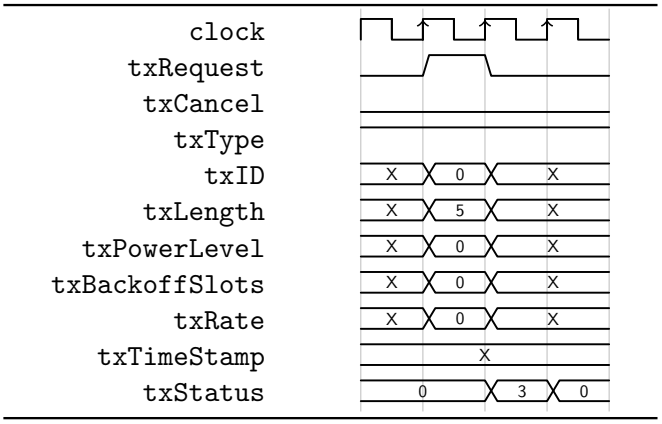

Figure 5.10: Failed real-time transmission timing diagram

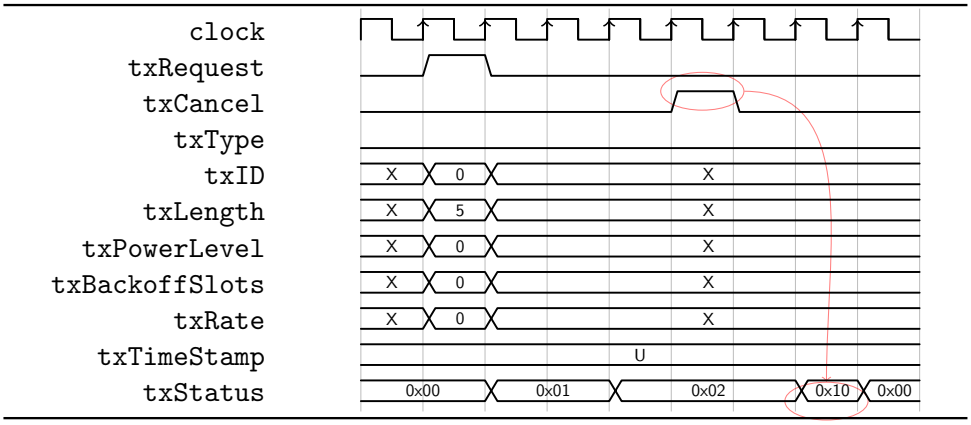

Figure 5.11: Cancelled transmission timing diagram

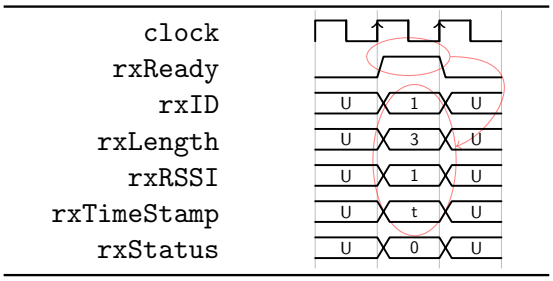

Figure 5.12: Successful reception timing diagram

5.1.7 Packet reception

The reception of a packet is completely automated and requires no external intervention. Under normal circumstances, depicted in fig. 5.12, when an incoming packet is detected, a memory slot is allocated to store the received data in it and all the necessary information relative to the received packet is reported and validated with `rxReady`. If the reception was successful, as signalled by the `rxStatus` signal, the used memory slot must be released after processing the received data. Failure to do so will result in a full memory after a few receptions and inability to store any more packets until memory is available. The exact meaning of each of the interface signals and the possible error conditions follow.

- `rxReady` signals the reception of a new packet. The other `rx` signals are only valid while `rxReady` is asserted. For each received packet this signal is active during one clock cycle.
- `rxID` is the identifier of the slot allocated to store the received data. Under some circumstances, reported in `rxStatus`, this identifier is not valid, in which no data relative to this reception is available and no memory release should be executed. **??** provides further details on the matter.
- `rxLength` is the size, in octects of the received packet. The size already includes the FCS, which are the 4 last octets. A value of 0 is possible whenever an error occurs and means that the size of the received packet is not known.
- `rxRSSI` is the signal strength indicator value as defined by the standard. This value ranges from 0 to 0 and its units are dBm. It is always valid.
- `rxTimeStamp` is the exact moment at which the first pulse of energy arrived at the antenna according to `RTC`. It is always valid.

When an error occurs, its source is reported via the `rxStatus` signal pictured in fig. 5.13. Two error conditions are distinguished and may occur at the same time. The first is the inability to correctly decode the received packet and is signalled via the `RXERR` bit. The other is the unavailability of memory to store the received data, signalled via the `NOMEM` bit. When an `RXERR` error occurs, the exact nature of the error is available at the 4 MSb of `rxStatus`. Some of the conditions that lead to this type of error are detected before a memory slot is allocated to store the incoming data, in which cases `rxID` is not valid and no memory release needs to be done. The bit `IDINV` is used to report this situation. Table 5.2 describes a few possible `rxStatus` values.

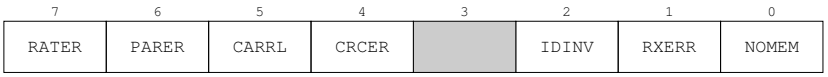

Figure 5.13: `rxStatus` fields

value	meaning
0x00	packet received and stored with success
0x05	packet received with success but no memory is available
0x12	packet received and stored, but CRC error detected
0x27	packet received with carrier lost error and no memory was available
0x46	packet received with rate error and no memory slot was allocated

Table 5.2: `rxStatus` signal meaning

5.1.8 Channel changing

Channel configuration is an operation that tunes the transceiver to the requested centre frequency. The only input parameter is the channel number. In order to avoid interfering with any ongoing operation, the channel changing is always postponed to a time when the medium is regarded as free. This behaviour is exposed in the interface via `configChannelStatus`. A request may be cancelled using the `configChannelCancel` signal.

- `configChannelRequest` requests the change of the configured channel to `configChannelValue`. If a request is issued before a previous one completed it is ignored. Figure 5.15 shows the operation details of this signal.

- `configChannelCancel` cancels an ongoing channel configuration request. Cancelling is only possible if the request is blocked. If the configuration is being performed at the moment, this signal has no effect. On the other hand, if the configuration is currently blocked it will be cancelled during the next clock cycle. Figure 5.16 shows the timing diagram of a cancellation.

- `configChannelValue` is the number of the channel to be configured. The numbering scheme follows the standard definition as shown in table 5.3.

- `configChannelStatus` reports the current state of the channel configuration operation. It is a bit field as shown in fig. 5.14. `CCCPL` is asserted when the operation completes with success, `CCCAN` is asserted during the operation cancelation, `CCBLK` is asserted whenever the operation is blocked and `CCONG` is asserted when the channel configuration is really taking place. When `CCONG` is asserted the operation can no longer be cancelled.

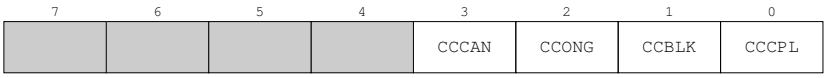

Figure 5.14: `configChannelStatus` fields

configChannelValue	centre frequency (MHz)
172	5860
174	5870
176	5880
178	5890
180	5900
182	5910
184	5920

Table 5.3: `configChannelValue` and central channel frequency match

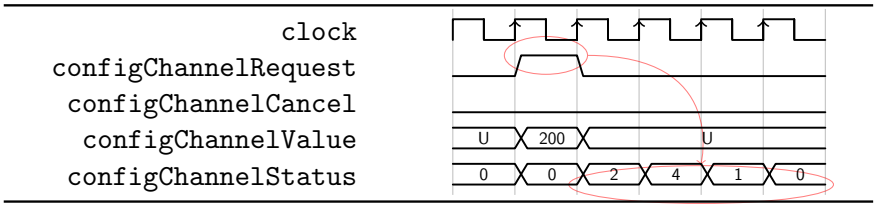

Figure 5.15: `configChannelRequest` operation details

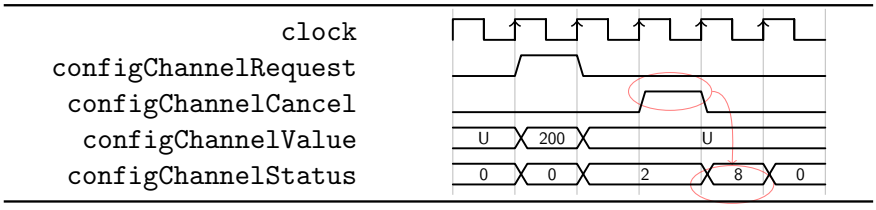

Figure 5.16: `configChannelCancel` operation details

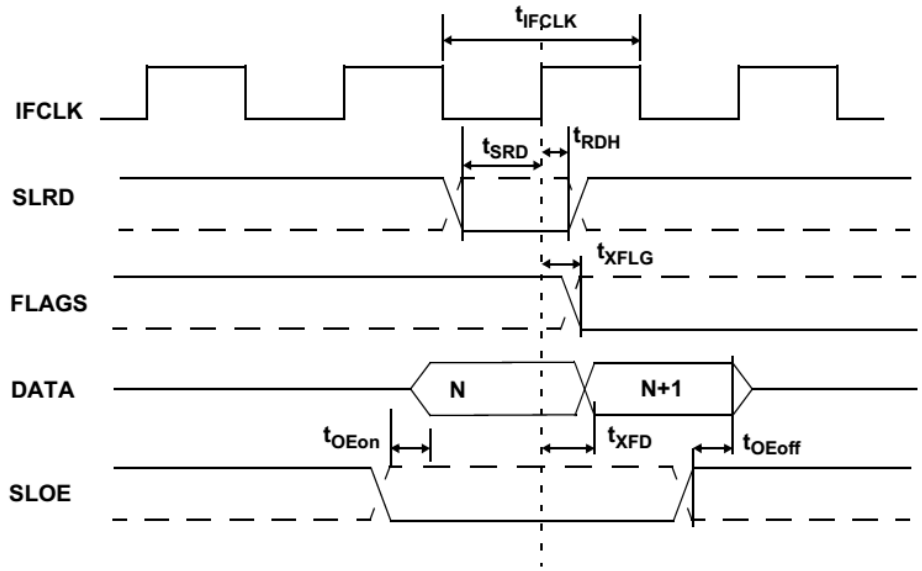

Figure 5.17: Timing diagram for slave FIFO read operation

5.2 USB Connection

The USB communication system responsible for providing a deterministic link between the LMAC and UMAC and also multiple radio support requires de implementation of three different components: the firmware that configures the FX2LP controller, the FPGA modules and the device driver that runs on the computer. It is also important to conduct a thorough timing analysis of the FX2LP module in order to ensure the correct operation of the interface signals.

5.2.1 FX2LP timming constraints

The slave FIFO interface has some timing constraints that will affect the development of the FPGA module that interacts with it. This section provides a summary of the specification provided in the datasheet and contains only the relevant information for this work.

When accessing a FIFO for reading, the associated constraints, shown in fig. 5.17 and described in table 5.4, lead to two important aspects to take into account in the design of a controller that correctly drives the interface signals. First of all, the `SLRD` setup time is very high, when compared to the minimum clock period, and it is very unlikely that the FPGA could generate such signal for non trivial designs. Since this is a worst case scenario and a read operation is executed every time this signal is asserted during a rising clock edge, introducing a wait state is not a possible solution, because it could introduce erratic behaviour. The only possible solution is to lower the interface operating frequency to an acceptable value. The hold time of the `FD` bus after the disabling of `SLOE` is not zero. This requires the introduction of a wait state after the disabling of `SLOE` to avoid electrical collisions.

The FIFO write interface, shown in fig. 5.18 and described in table 5.5, is very straightforward to implement and does not require any special considerations.

After careful analysis of the FX2LP datasheet, one more more signal timing information was considered of of crucial importance on the design of the controller, which is the `FIFOADR`

Parameter	Description	Min	Max	Typ		Unit
				Min	Max	
t_{IFCLK}	IFCLK period	20.83	–	–	–	ns
t_{SRD}	SLRD to clock setup time	18.7	–	–	–	ns
t_{RDH}	Clock to SLRD hold time	0	–	–	–	ns
t_{OEon}	SLOE turn on to FIFO data valid	–	10.5	–	–	ns
t_{OEoff}	SLOE turn off to FIFO data hold	–	10.5	–	–	ns
t_{XFLG}	Clock to FLAGS output propagation delay	–	9.5	–	–	ns
t_{XFD}	Clock to FIFO data output propagation delay	–	11	–	–	ns
t_{IFCLKR}	IFCLK rise time	–	–	–	900	ps
t_{IFCLKF}	IFCLK fall time	–	–	–	900	ps
$t_{IFCLKOD}$	IFCLK Output duty cycle	–	–	49	51	%
t_{IFCLKJ}	IFCLK jitter peak to peak	–	–	–	300	ps

Table 5.4: Description of timing diagram information for slave FIFO read operation

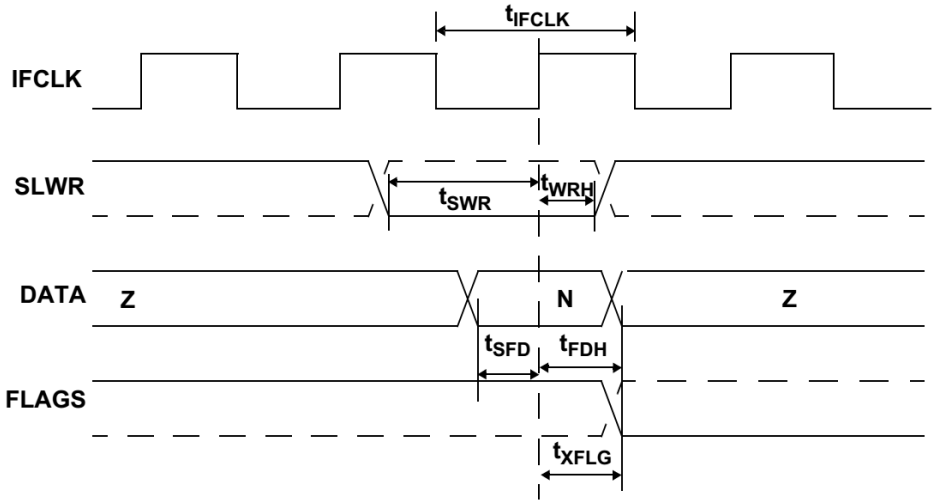

Figure 5.18: Timing diagram for slave FIFO write operation

Parameter	Description	Min	Max	Unit
t_{IFCLK}	IFCLK period	20.83	–	ns
t_{SWR}	SLWR to clock setup time	10.4	–	ns
t_{WRH}	Clock to SLWR hold time	0	–	ns
t_{SFD}	FIFO data to clock setup time	9.2	–	ns
t_{FDH}	Clock to FIFO data hold time	0	–	ns
t_{XFLG}	Clock to FLAGS output propagation time	–	9.5	ns

Table 5.5: Description of timing diagram information for slave FIFO write operation

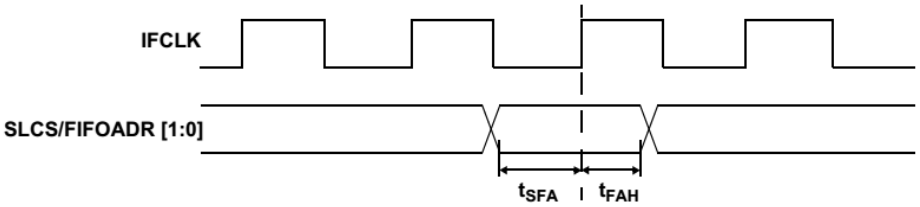

Figure 5.19: Timing diagram for slave FIFO `FIFOADR` signal

Parameter	Description	Min	Max	Unit
t_{IFCLK}	Interface clock period	20.83	200	ns
t_{SFA}	FIFOADR[1:0] to clock setup time	25	–	ns
t_{FAH}	Clock to FIFOADR[1:0] hold time	10	–	ns

Table 5.6: Description of timing diagram information for slave FIFO `FIFOADR` signal

signal. As observed in fig. 5.19, which is described in table 5.6, the hold time of this signal is greater than the minimum operation period. If the internal clock generator is used, which is a good solution to avoid having to generate a clock using FPGA resources, two clocks frequencies are available: 48MHz and 30Mhz, which correspond to 20.83ns and 33.33ns respectively. In order to correctly generate this signal using an FPGA there are two solutions: either introduce a wait state before driving the remaining interface signals or lower the operating frequency of the interface. The simplest solution was deemed to be the dropping of the operating frequency to 30MHz. Using this value the FPGA should have no problem in generating a signal with the appropriate timing. This solution also solves the problem previously identified with the `SLRD` signal setup time.

5.2.2 FX2LP firmware

The firmware developed for the FX2LP is very simply a sequence of commands that configure the slave FIFO interface to operate as required. The code provided by the FPGA module provider [Ele], was adapted to the needs of this work in the following ways:

- activate all endpoints according to table 5.7
- activate double 512 byte buffering in each endpoint
- configure the flag pins to provide full information for the `IN` type endpoints and empty information for the `OUT` ones, as shown in table 5.7
- use internal clock generator to driver `IFCLK` at 30MHz

After this adaptations, the code was compiled using the open source sdcc compiler and loaded to the chip using the fxload program provided along with the libusb software [lib]. The firmware is loaded to the device volatile memory, therefore must be downloaded every time the platform is powered up. This process was automated using the standard udev tools of the Linux operating system.

Endpoint number	Type	Direction	Associated flag
2	OUT	Interrupt	`FLAGA`
4	OUT	Bulk	`FLAGB`
6	IN	Interrupt	`FLAGC`
8	IN	Bulk	`FLAGD`

Table 5.7: FX2LP endpoint configuration

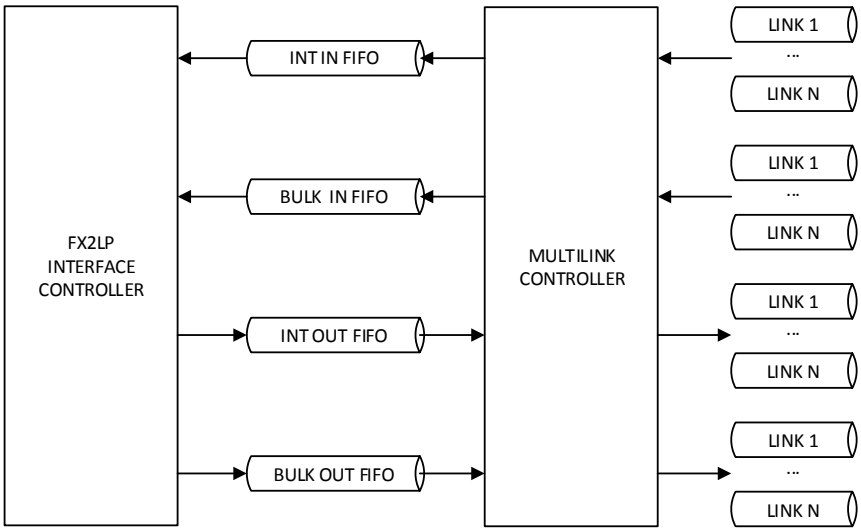

Figure 5.20: FPGA module for the USB controller

5.2.3 FPGA module

A controller module following the interface shown in section 4.4 and the recommendations of the previous subsections was developed. This controller is made of two components, one that interfaces the FX2LP and the other that implements the multiplexing mechanism to allow multiple independent user, as exhibited in fig. 5.20. The components are interconnected using dual clock FIFOs to cross from the clock domain of the FX2LP interface to the internal FPGA clock.

FX2LP controller

This module is responsible for correctly driving the FX2LP slave interface signals, and operate the dual clock FIFOs used to cross from the clock domain used to interface the FX2LP to the one used in rest of the FPGA. The interface with the FX2LP only provides one data bus, so the reading and writing of data in different endpoints must be made sequentially. Since there are four FIFOs that need access to this bus, the access must be made under some sort of scheduling mechanism. The scheduling mechanism implemented by this module is fixed priorities, where the priorities are, from lowest to highest: `INT OUT, INT IN, BULK OUT, BULK IN`. The implementation is based on a finite state machine and datapath (FSMD)

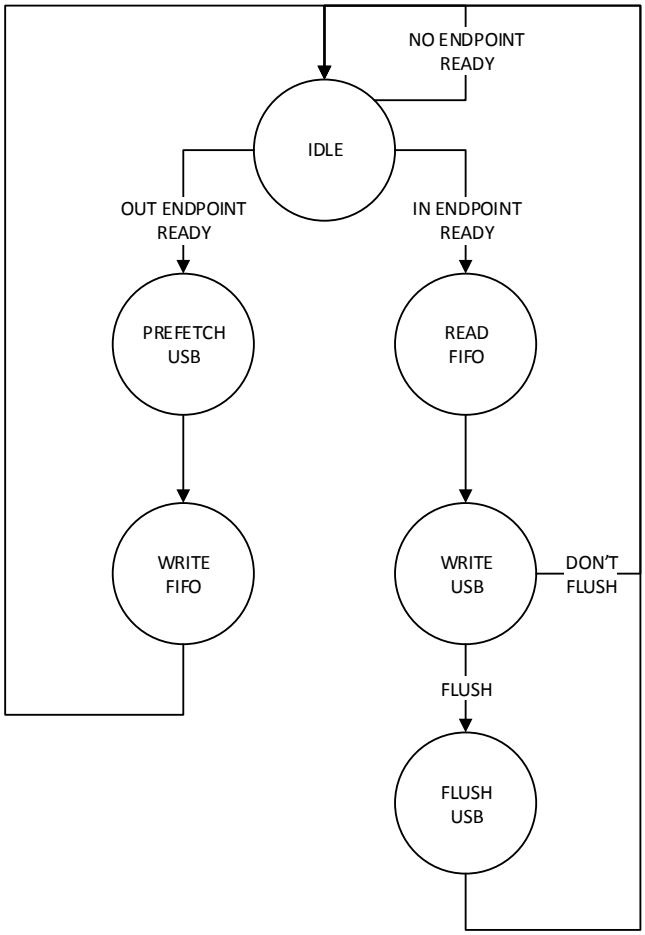

Figure 5.21: FX2LP controller state machine

design, as is usual for FPGA based controllers. The state machine transition diagram is shown in fig. 5.21.

Multilink controller

After crossing from the domain of the FX2LP signals to a regular FIFO interface, it is necessary to implement the multiplexing/demultiplexing scheme presented in section 4.4. After this operation, a set of FIFOs will be provided for each link to use independently.

5.2.4 Software module

The software developed allows the use of this communication system by multiple independent users by means of a shared memory region. This region is managed by a process that schedules the transmissions of each user in order to guarantee fair access to the bus and does the multiplexing/demultiplexing task required by the underlying transport protocol already

described. Each one of the links is exported to the application as a set of FIFOs, which can be accessed via a simple software interface.

5.3 Device Driver

This work is concerned with implementation of deterministic MAC schemes in software with appropriate hardware support, and in order for the complete system to be deterministic, operations must be complete in a deterministic amount of time. Both the hardware and the communication system already ensure deterministic behaviour, so, in order to have a complete deterministic system, the driver must take care to ensure completion of the requested operations in a bounded amount of time. The Linux operating system does not in fact provide this kind of guarantees, but if certain conditions are followed, a good approximation can be obtained:

- use the FIFO scheduler
- run processes with maximum priority
- use `mlockall` to ensure process memory region is always in RAM
- use shared memory for inter process communication

The implementation of the device driver is based on a shared memory region where the already presented calls operate. This region is maintained by a process that forwards the application requests to the device via the USB communication system. Internally, this piece of software simply follows the described architecture and uses a different thread for each one of the tasks shown in fig. 4.12. Synchronization between each one of the different calls and the handling thread is performed using semaphores which are created inside the shared memory.

Chapter 6

Validation

To test the implemented solution, two different stages where devised. First, the USB based communication system was tested in order to assess its adequacy for this application. Then, the higher level functions were tested to verify that the system behaves adequately, as a whole.

6.1 MultiLink

The goal of the tests conducted to MultiLink is to verify that the complete USB communication system works as expected. In order for the whole system do work, not only the implementation must be correct, but also the assumptions about the behaviour of the USB controllers, both host and device. While this implementation is not able to modify the host controller, it may be possible to adjust the behaviour of the device controller if required. The tests aimed at measuring the latency and throughput of the bounded latency channels.

6.1.1 Test setup

In order to correctly assess a system performance it is usually required that the measuring entity be distinct from the system under test. While it is true that for micro-controller based devices it is not entirely possible to isolate two pieces of code running on a single device, the same does not apply to an FPGA based system. For this reason, the FPGA is tasked with not only implementing the `MultiLink` controller but also the necessary mechanisms to assess its performance. The distributed nature of the FPGA design guarantees that no interference between the two ever occurs.

The testing procedure requires three elements:

- the `MultiLink` host: RaspberryPi
- the `MultiLink` device: FPGA
- external monitoring device: PC

The `MultiLink` host, in this case a RaspberryPi is responsible for running the test application to apply the intended stimulus to the system, as well as the software layers of `MultiLink`. The `MultiLink` device is responsible for running the device side of the implementation and perform all the required measurements. Although the IT2S platform could

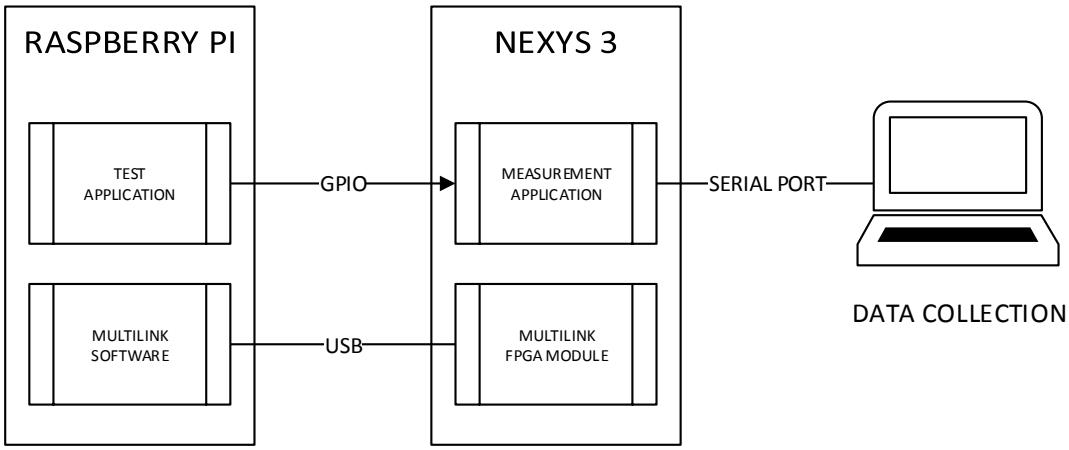

Figure 6.1: Test setup for the MultiLink implementation

be used as a device, it was decided that a Nexys 3 prototyping board should be used, not only because it provides greater flexibility in terms of I/O solutions to interact with external systems, but also to test the portability of the designed solution. This board uses the exact same USB controller that the platform, so the firmware can be used without changes. The only required modifications concern the pin attribution in the FPGA design, which naturally have to match the new board. Both systems are connected using a GPIO pin that is used to trigger a measurement. The response time of this type of interface is much lower than the times intended to be measured during this tests. At last, an external computer connected to the FPGA module via a serial port is used to collect the data gathered during the tests. The whole setup can be observed in fig. 6.1.

6.1.2 Latency measurement

The application software generates a packet with a period big enough to ensure that the bus is within its working bandwidth limits. The parameter that is being tested is latency, not throughput. Before issuing the call that transmits the information, the host asserts the GPIO pin connecting it to the FPGA. At this moment, the FPGA measuring system starts a counter that will allow the measurement of the time it takes for the packet to reach the FPGA fabric. When the first byte of the packet arrives at the FPGA, the amount of measured time is transferred to the data collecting PC for analysis. A similar test was also performed in the inverse direction, where the FPGA signals the measurement system to start the time counting and then proceeds to send a packet to the PC. When the packet is received by the PC, the GPIO pin is asserted and the measurement system reports the measure time to the data collecting PC. The procedure was repeated 100000 times in each direction, and the obtained results can be observed in fig. 6.2.

The obtained results show that the mean latency value is around 0.3 ms for the out endpoint but much lower for the in endpoint. This can be explained by the fact that during an in transmission, the host is already expecting the data before it is sent by the device,

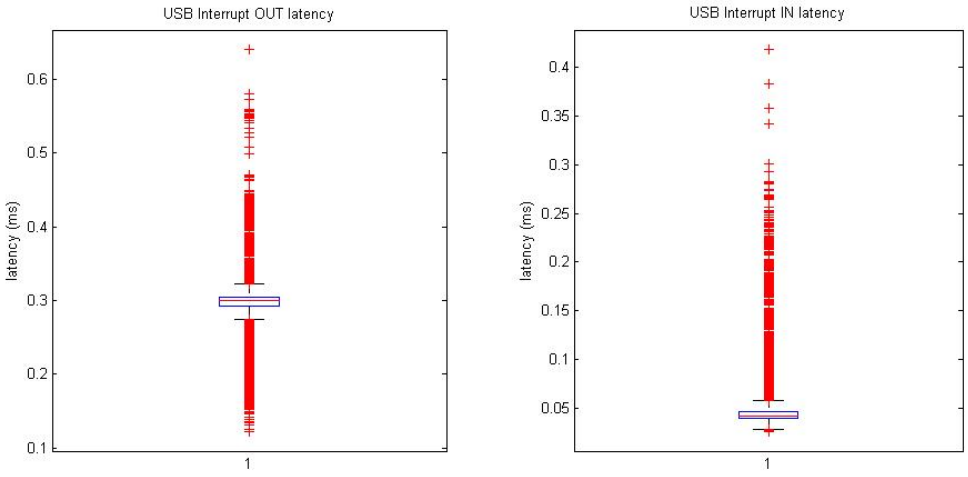

Figure 6.2: USB latency measurement results architecture

as is required by the USB specification, therefore there is less code to be executed after the transmission starts, as opposed to an out transfer. No occurrences were registered with a latency value exceeding 1 ms.

6.1.3 Throughput measurement

The procedure use to measure the throughput of the system is somewhat similar to the one used to measure the latency, but in this scenario, instead of sending the packet spaced in time, the packets are transmitted at the maximum possible rate, in order to keep the bus occupied at all times. In this case, the FPGA still collects the timestamps of the received packets, but for the purpose of calculating the bandwidth of the channel, so it is not necessary to synchronize the measurement system with the data generating application, because only relative time is of importance here. The data collecting tool gathers one timestamp value for each received packet. A similar procedure is used to measure the throughput from the FPGA to the PC, but in this case, the timestamp is stored when the packet is successfully transmitted. A packet can only be successfully transmitted after the buffers have enough space for it, meaning that any previous packet occupying the said buffers was already transmitted. While this could lead to false burst values, a long term analysis should proved to yield the correct results. The collected variable for analysis is the time between each transmission, or the transmission period.

The obtained values for periods are observed in fig. 6.3. One interesting result if the fact that the mean period is 1 ms, yielding 1000 packets per second. This is not the expected value, as the USB specification states that interrupt type endpoints should be able to provide up to 8000 transfers a second. After some preliminary tests with a different system running the MultiLink software, it was concluded that the RaspberryPi is to blame for the issue. The USB implementation of this embedded PC is not according to the standard specification and only provides one interrupt transfer per millisecond. Since this device is part of the platform, and it is not the objective of this work to modify the said platform, this value will be regarded as a system constraint, and the guarantees provided by the system adjusted

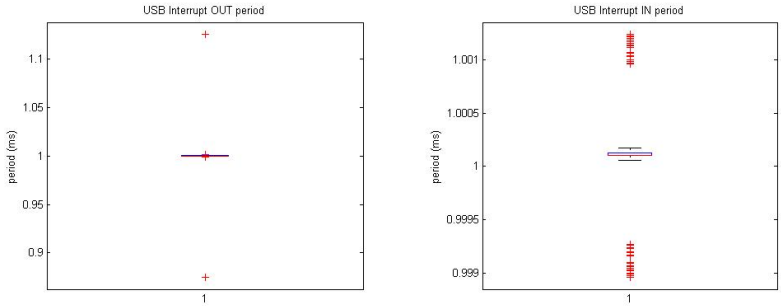

Figure 6.3: USB period measurement results architecture

accordingly. It should be noted however that the performance should be still quite sufficient for most vehicular communication applications.

6.2 Deterministic MAC

In order to test the mechanisms that allow the implementation of a deterministic MAC, one also needs two devices. One to transmit the information and one to receive and make measurements. This tests will leverage the two radio solution provided by the IT2S platform and the fact the both radios share the same RTC provider, allowing for time measurements to be performed without concern for misaligned time frames.

6.2.1 Time triggered packet transmission

The testing of the time triggered packet transmission was performed by simply requesting to send a packet at a specific moment and measuring the timestamp provided by the receiver. Once again, since the receiver and transmitter share the same RTC all involved times are synchronized. The obtained results show that all transmissions were started with 1 us of the specified instant. Although this test was conducted under ideal conditions, without interference from external units, due to the fact that it is possible to disable the collision avoidance mechanisms, it can be regarded as a realistic scenario.

6.2.2 Sensitivity adjustment

The testing of the sensitivity adjustment mechanism was performed with two distinct platforms. In this case, no time measurements are involved, therefore there is not problem with the possibly differing RTC.

The procedure used in this measurement was to place the platforms at several known distances, always with line of sight (LOS) to avoid interferences caused by obstacles and for each position make several transmissions of a set of frames with a fixed power, using different receiver sensitivity for each set of transmitted frames. This gathered data is the packet error rate (PER) value for each distance and receiver sensitivity. The obtained results are shown in fig. 6.4.

Although the data follows the expected trend of increasing PER with decreasing sensitivity, for the same distances, and the transmission power was kept at the minimum value, the

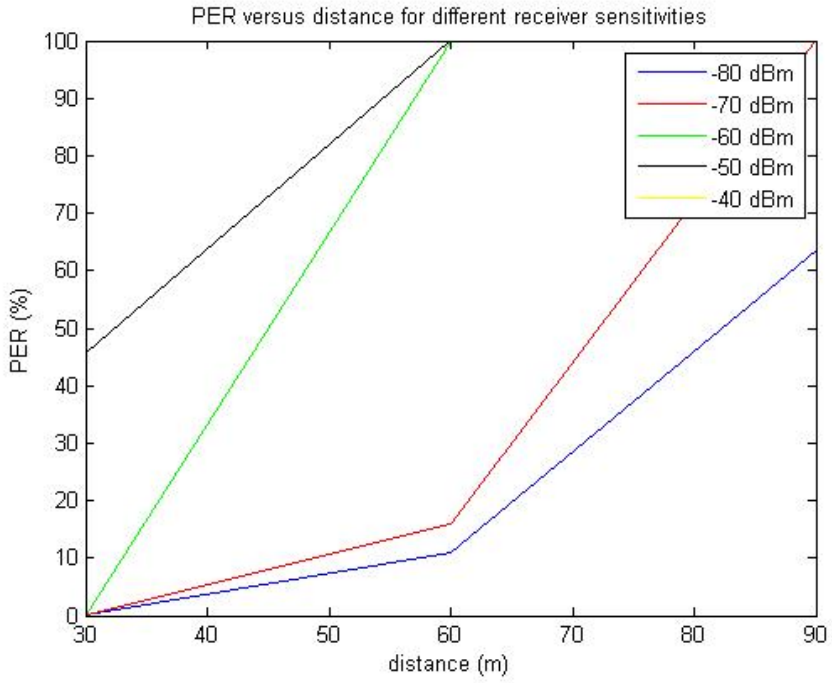

Figure 6.4: Sensitivity adjustment test results

measured PER is higher than would be expected for the distances used during the test. This factor could be due to some terrain feature in the test scenario or some reflections caused by surrounding buildings. Nevertheless, as already stated, the results show the expected trend and any distance limiting factor is affecting all test cases in a similar way.

Chapter 7

Conclusions and Future Work

The proposed devise was successfully implemented and shown to be working according to the required expectations.

A by product of this work was also the development of a generic USB communication system that allows the transfer of data between a PC and an FPGA with certain latency and throughput guarantees, that can be simultaneously used by multiple users, both on the PC and the FPGA sides. It can be reused on further works, not only using this platform, but any platform integrating an FX2LP device connected to the FPGA, as was shown in the tests performed using a Nexys 3 board.

Future work should include the implementation of deterministic MAC schemes using the develop platform and conducting field tests with the said schemes in order to assess their performance under real conditions. A comparison can later be made with the simulated environment results in order to verify the correctness of the assumptions.

Some work remains to be done on the time synchronization module. Currently the Pulse Per Second (PPS) signal provided by the GPS module is being used to provide a time value that is always relative to the last second. While this proved enough to conduct the tests, it is not enough to use in real scenarios, due to the absence of filtering mechanisms that allow a time value to be kept accurate during momentary losses of GPS signal.

Bibliography

[ACM+13] A. Autolitano, C. Campolo, A. Molinaro, R.M. Scopigno, and A. Vesco. An insight into decentralized congestion control techniques for vanets from etsi ts 102 687 v1.1.1. In *Wireless Days (WD), 2013 IFIP*, pages 1–6, Nov 2013.

[BUSB09] Katrin Bilstrup, Elisabeth Uhlemann, Erik G. Ström, and Urban Bilstrup. On the ability of the 802.11p mac method and stdma to support real-time vehicle-to-vehicle communication. *EURASIP J. Wirel. Commun. Netw.*, 2009:5:1–5:13, January 2009.

[COFM14] Cristóvão Cruz, Arnaldo Oliveira, Joaquim Ferreira, and João Matos. Implementing deterministic vehicular communications: Rationale and challenges. In *6th IEEE Vehicular Networking Conference (VNC 2014)*, Paderborn, Germany, December 2014. IEEE.

[dA13] João Miguel Pereira de Almeida. Plataforma multi-rádio para comunicações veiculares dsrc 5.9 ghz. Master's thesis, Universidade de Aveiro, 2013.

[dS11] Manuel José Alves Ventura da Silva. Co-projecto em fpga da mac ieee 802.11p para comunicações veiculares. Master's thesis, Universidade de Aveiro, 2011.

[Ele] Trenz Electronic. Trenz TE-USB Suite. `https://github.com/Trenz-Electronic/TE-USB-Suite`. Accessed: 2014-12-11.

[ESG13] D. Eckhoff, N. Sofra, and R. German. A performance study of cooperative awareness in etsi its g5 and ieee wave. In *Wireless On-demand Network Systems and Services (WONS), 2013 10th Annual Conference on*, pages 196–200, March 2013.

[ETS] ETSI. ETSI TS 102 687 V1.1.1; Intelligent Transport Systems (ITS); Decentralized Congestion Control Mechanisms for Intelligent Transport Systems operating in the 5 GHz range; Access layer part. `http://www.etsi.org/deliver/etsi_ts/102600_102699/102687/01.01.01_60/ts_102687v010101p.pdf`. Accessed: 2014-12-11.

[FOAC13] Joaquim Ferreira, Arnaldo Oliveira, João Almeida, and Cristóvão Cruz. Fail silent road side unit for vehicular communications. In *ASCoMS@SAFECOMP*, 2013.

[Hea14] Headway. HEADWAY - Connecting vehicles and highways. `http://www.brisainovacao.pt/en/innovation/projects/headway`, 2014. Accessed: 2014-12-11.

[HG11] Bin Hu and Hamid Gharavi. A joint vehicle-vehicle/vehicle-roadside communication protocol for highway traffic safety. *International Journal of Vehicular Technology*, 2011, 2011.

[ICS14] ICSI. Intelligent Cooperative Sensing for Improved Traffic Efficiency. `www.ict-icsi.eu/description.html`, 2014. Accessed: 2014-12-11.

[IEE12] IEEE. 802.11-2012 - IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, 2012.

[lib] libusb. fxload. `https://github.com/libusb/libusb/blob/master/examples/fxload.c`. Accessed: 2014-12-11.

[MTC+10] F.J. Martinez, Chai-Keong Toh, J.-C. Cano, C.T. Calafate, and P. Manzoni. Emergency services in future intelligent transportation systems based on vehicular communication networks. *Intelligent Transportation Systems Magazine, IEEE*, 2(2):6–20, Summer 2010.

[Org13] World Health Organization. *Global status report on road safety 2013 : supporting a decade of action.* World Health Organization Geneva, 2013.

[PLFE+09] Panos Papadimitratos, A La Fortelle, Knut Evenssen, Roberto Brignolo, and Stefano Cosenza. Vehicular communication systems: Enabling technologies, applications, and future outlook on intelligent transportation. *Communications Magazine, IEEE*, 47(11):84–95, 2009.

[Sem] Cypress Semiconductor. Ez-usb technical reference manual. Document 001-13670 Rev. D.

[SWL+12] Sundar Subramanian, Marc Werner, Shihuan Liu, Jubin Jose, Radu Lupoaie, and Xinzhou Wu. Congestion control for vehicular safety: Synchronous and asynchronous mac algorithms. In *Proceedings of the Ninth ACM International Workshop on Vehicular Inter-networking, Systems, and Applications*, VANET '12, pages 63–72, New York, NY, USA, 2012. ACM.

[TJJ13] Meireles T., Fonseca J., and Ferreira J. Vehicular Flexible Time-Triggered Protocol (V-FTT). Technical Report 1, Instituto de Telecomunicações - Embedded Systems Group, March 2013.