Implementation and Test of a DSRC Prototype on OpenAirInterface SDR Platform

Philippe Agostini*, Raymond Knopp†, Jérôme Härri† and Nathalie Haziza*
*Thales Communications & Security, Gennevilliers, France
†Mobile Communications Department, EURECOM, France

***Abstract*—This paper presents OpenAirITS, an open-source Software-Defined Radio platform for DSRC (802.11p) technology. We extended the Linux 802.11 subsystem, developed a soft-modem and a dedicated driver for the OpenAirInterface Express-MIMO FPGA board. The low-layer PHY functions of DSRC have not been coded on the chipset, but instead as a soft-modem, which allows the SDR platform to be easily modified according to particular experimental objectives. The RF front-end and Express-MIMO board are reconfigurable to allow a wide range of options. In this paper, we configure the prototype for a 5Mhz channel bandwidth at at 800MHz, and provide key performance metrics of the soft-modem, as well as the DSRC protocol stack.**

***Index Terms*—IEEE 802.11p, DSRC, prototype, Linux 802.11 subsystem, software defined radio, OpenAirInterface, vehicular communications, ITS.**

I. Introduction

The perspective of innovative safety and non-safety Intelligent Transportation Systems (ITS) applications motivated the increasing R&D efforts over the past few years to provide an efficient and reliable dedicated communication system for vehicular environments based on DSRC/IEEE 802.11 [1]. In the development path, after design and simulation-based evaluations, prototyping and field operational tests are the last step before market introduction. It is also the most challenging and crucial phase, as the developed solutions face real constraints and challenging vehicular environments, that simulations cannot reproduce. Several platforms (NEC LinkBird-MX [2], Denso WSU [3], Cohda MK2 [4],..) are available, which are also currently under test within large FOT projects like Drive C2X [5].

One limitation from the few available platforms is that they are developed for a particular configuration and bound to hardware/chipset constraints. This reduces their ability to keep up with the dynamic standardization process currently being conducted. Also, it has also been observed that major improvements could be provided to vehicular communications by adjusting or modifying the DSRC lower layers functionalities or parameters. Unfortunately, most of the available experimental platforms either do not release the source code, or have lower layer functions directly coded on the chipset. These aspects either limit or even block the rapid development and test of innovative techniques to improve vehicular communications.

This work has been partially supported by the French ANR for the PLATA project and the French-German cooperation (DEUFRAKO), and partially by the FP7 NoE Newcom#. EURECOM acknowledges the support of its industrial members: BMW Group Research & Technology, IABG, Monaco Telecom, Orange, SAP, SFR, ST Microelectronics, Swisscom, Symantec.

In this paper, we present the implementation and test of OpenAirITS[1], an open-source hardware/SDR DSRC prototype on the OpenAirInterface platform. The OpenAirInterface flexible *Lime RF* front-end and *ExpressMIMO* FPGA board support static and dynamic frequency changes between 400Mhz and 6GHz, the lower IEEE 802.11 PHY functionalities have been implemented as a soft-modem, the Linux 802.11 subsystem has been modified to support the OCB function[2], and the prototype is made available to the Linux IP subsystem as a virtual interface. This paper describes the design methodologies and the test procedures to evaluate the prototype's key performance metrics. We illustrate the experimental capabilities of the DSRC prototype by selecting an experimental 5MHz band at 800MHz[3], but the platform is fully capable of supporting 10MHz or 20MHz band at 5.9GHz. The platform being open-source, we provide a tool to the vehicular communication and flexible radio community for fast prototyping and experimental validations of DSRC components, connectivity, as well as ITS applications. Also, benefiting from the flexible *Lime RF* front-end, the prototype may be adapted to the different worldwide standards (US, EU, Japan), as well as experimental frequencies or cognitive approaches.

The rest of the paper is organized as follows. Section II positions this work within related work in DSRC prototyping, while Section III depicts the general architecture of the OpenAirITS prototype. In Section IV, we describe the various developed software components, whereas in Section V and Section VI, we present experimental tests respectively on a single device or by connecting two devices with a coaxial cable. Finally, Section VII discusses current and future developments of the prototype.

II. Related Work

Prototyping communication systems for ITS applications is mostly based on software development kits (e.g. [2], [6]–[8]) modeling the upper-layer functions built on top of communication platforms (e.g. [2]–[4]). This approach is also followed by major FOT projects such as Drive C2X [5]. But when the lower layers also need to be experimentally enhanced, the

[1]The platform code is available at http://www.openairinterface.org/

[2]The OCB function in IEEE 802.11-2012 [1] represents the MAC functionalities previously available in the IEEE 802.11p amendment.

[3]Within the PLATA project, a 5Mhz frequency band at 800MHz has been selected first to use the same transmit antenna and same configuration for the RF front-end as the OpenAirInterface LTE transceiver collocated on the same chipset, but also for its resilience to fading for safety-related communication.

few available open-source platforms usually provide limited 802.11p OFDM functions [9].The prototype described in this paper is based on an open-source software/hardware platform OpenAirInterface, and does not only provide a fully functional OFDM 802.11p but also a modified 802.11p MAC stack. The choice of OpenAirInterface to implement a DSRC hardware/software prototype is also justified by the compatibility with the OpenAirInterface's LTE-A protocol stack.

III. Software Architecture

Figure 1 illustrates the open-source SDR architecture of our IEEE 802.11p protocol stack. It is composed of three blocks. The upper block contains an extension for IEEE 802.11p of the Linux kernel 802.11 subsystem. This subsystem is composed of *nl80211*, a netlink configuration interface for user-space applications, *cfg80211* which is the Linux wireless configuration interface bridging user-space and drivers and *mac80211* which offers a framework for driver developers writing soft-MAC wireless devices. The *mac80211* subsystem is the Linux stack for IEEE 802.11. Due to tight real time constraints, the 802.11 subsystem only implements basic parts of the IEEE 802.11 standard, the other parts being usually the responsibility of manufacturer chipsets.

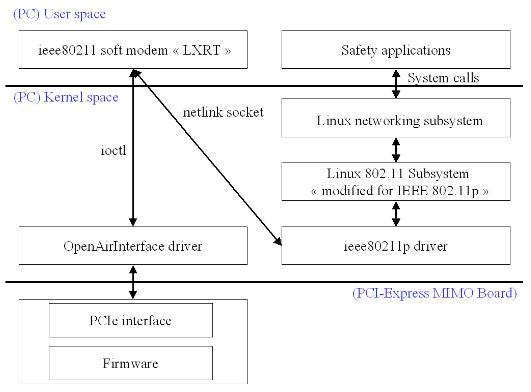


Fig. 1. IEEE 802.11p protocol stack

The second block is the *IEEE 802.11p driver*, which bridges the Linux 802.11 subsystem and the hardware. One major difference with standard architecture for 802.11 systems, is that the IEEE 802.11p driver does not link to the chipset directly, but rather to a *soft-modem* via netlink sockets. We chose this architecture for flexibility in the development and configuration of the low layer functionalities of the IEEE 802.11p stack and to ease the reconfigurability of our radio. The *soft-modem* is the placeholder of all the functionalities of IEEE 802.11p physical layer. Being located out of the chipset, it is totally accessible and reconfigurable. The soft-modem is finally connected to the hardware via an IOCTL link and a dedicated OpenAirInterface driver, which composes the last block.

Figure 2 and Figure 3 illustrate the software architecture for the TX and RX paths, and depict the various functions and data structures employed by the IEEE 802.11p protocol stack in kernel space. We will cover them with more details in the next section.

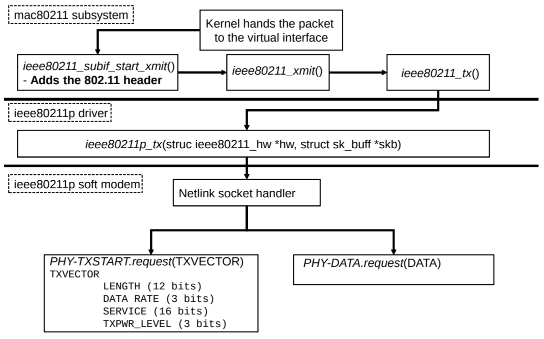


Fig. 2. IEEE 802.11p TX Path

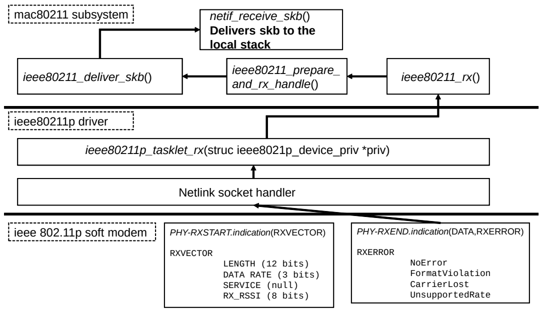


Fig. 3. IEEE 802.11p RX Path

IV. Software Components

We describe in this section some details of the software components in each block.

A. Linux 802.11 subsystem

According to the IEEE 802.11-2012 standard [1], when the flag *dot11OCBActivated* is set to *true*, the Linux 802.11 subsystem may accept communications conducted outside of the context of a basic service set (OCB mode), and as such, may bypass the scanning, authentication and association steps of the IEEE 802.11 state machine. As this mode is strongly related to the capability of the IEEE 802.11 chipset (frequency, half-rate etc..), the driver should also indicate it to the subsystem with a flag *IEEE80211_HW_DOT11OCB_SUPPORTED* to *1*.

We modified the Linux 802.11 subsystem accordingly, and also added missing ITS frequency bands 5GHz and 800MHz to the data structures *nl80211_band* and *ieee80211_band*. When both *dot11OCBActivated* and *IEEE80211_HW_DOT11OCB_-SUPPORTED* are set, the *BSSID* in the MAC header is replaced by the *wildcard BSSID*= $0xFFFF$ and delivered to the soft-modem on the TX path, and a packet containing a *wildcard BSSID* will be accepted without authentication or association and delivered on the RX path.

Broadcast, multicast and unicast may be used, but since safety-related low latency data exchanges are targeted, request to send (RTS), clear to send (CTS) as well as acknowledgments (ACK), fragmentation and QoS are not yet supported.

Parameter	Value	Linux 802.11
NL80211_IFTYPE	AD HOC	wiphy →if_modes
IEEE80211_BAND	800 MHz	wiphy →bands
dot11OCBActivated	true	wiphy
DOT11OCB_SUPPORTED	true	hw →flags

TABLE I
IEEE 802.11P DRIVER INIT CONFIGURATION OF THE 802.11 SUBSYSTEM

B. IEEE 802.11p driver

The *IEEE 802.11p driver* contains three basic routines: *INIT*, *TX* and *EXIT*. The *INIT* phase initializes the 802.11 subsystem and initiates a netlink socket calling a RX handler waiting for data from the soft-modem to transfer it to the 802.11 subsystem. Upon reception of data, the RX handler schedules a tasklet to handle the received frame and passes it to the Linux 802.11 subsystem. In the *TX* routine, the driver transfers data received from the 802.11 subsystem to the soft-modem via a netlink socket. The *EXIT* routines releases the RX handler, the tasklet and the netlink socket. Fig. 4 illustrates the basic functionalities and behavior of the IEEE 802.11p driver. It builds a bridge between the soft-modem and the Linux 802.11 subsystem, notably regarding statistics, and is also responsible for setting the interface type, the frequency band and the OCB mode as illustrated in Table I.

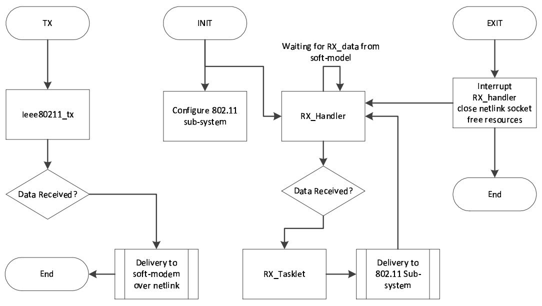

TX
INIT
EXIT
Waiting for RX_data from soft-model
Ieee80211_tx
Configure 802.11 sub-system
RX_Handler
Interrupt RX_handler close netlink socket free resources
Data Received?
Data Received?
End
End
Delivery to soft-modem over netlink
RX_Tasklet
Delivery to 802.11 Sub-system

Fig. 4. IEEE 802.11p driver flow chart

C. IEEE 802.11p Soft-modem

The key innovation of the soft-modem is to provide software defined instead of hardware-defined low-layer operations that are traditionally on a chipset. Having it software-defined and open-source makes it extensible and easily accessible for experimental design. The architecture of the soft-modem is depicted in Fig. 5, and contains channel monitoring as well as the block operation of an OFDM TX/RX.

The current version of the soft-modem has the full functionalities of the OFDM PHY layer, notably the mandatory and five other optional coding rates (BPSK 1/2, BPSK 3/4, QPSK 1/2, QPSK 3/4, 16QAM 1/2, and 16QAM 2/3) at the TX side, as well as energy detection and FCS check at the RX side. The MAC layer functionalities are limited, as contention-based access is not implemented yet.

From an interface perspective, the soft-modem provides the following open SAP: *PHY_TXSTART_request(TX_VECTOR)* and *PHY_DATA_request(DATA)* on the TX side, as well as *PHY_RXSTART_indicate(RX_VECTOR)* and *PHY_-RXSTART_indicate(DATA, RX_ERROR)*, where SAP parameters are indicated on Table II. As indicated, service quality (*Traffic Class*), transmit power and data rate may be specified, and represent the triplet for controlling congestion and QoS for IEEE 802.11p. On the RX side, statistics such as RSSI, as well as the used *Traffic Class*, transmit power and data rate may be forwarded to the 802.11 subsystem. In case of RX errors, the cause is indicated, such as invalid FCS.

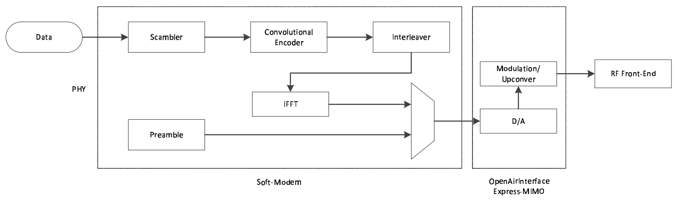


Fig. 5. Simplified schema of the design architecture of the soft-modem, and the interaction with the Express MIMO board and RF front-end.

TX_VECTOR		RX_VECTOR		RX_ERROR
LENGTH	12 bits	LENGTH	12 bits	NoError
DATA RATE	3 bits	DATA RATE	3 bits	FormatViolation
SERVICE	16 bits	SERVICE	null	CarrierLost
TXPWR_LEVEL	3 bits	RX_RSSI	3 bits	UnsupportedRate

TABLE II
SOFT-MODEM SAP PARAMETERS

V. VALIDATION

Before testing the prototype, we need to evaluate its conformity. To this objective, we first tested the soft-modem using an oscillator as a perfect OFDM traffic source, and tested the performance of the soft-modem in RX mode, while varying the SNR. We then mounted the 802.11p OCB interface, sent traffic from the soft-modem and intercepted replies to compute performance statistics.

Due to its reconfigurability, the SDR DSRC prototype may be tuned to various frequency bands or channel bandwidth. For these tests, and following the FOT specifications for the PROTON/PLATA project [10], we tuned it to the 800MHz frequency band at 5MHz channel bandwidth. We configured the soft-modem parameters accordingly (see Table III, and Table IV for the RF front-end capabilities).

A. IEEE 802.11p Soft-modem

We conducted three tests to evaluate the performance of the soft-modem and the OpenAirInterface FPGA board:

Parameters	5MHz	10 MHz	20 MHz
Symbol Duration	16 μs	8 μs	4 μs
Preamble Duration	64 μs	32 μs	16 μs
Slot time	21 μs	13 μs	9 μs
SIFS	64 μs	32 μs	16 μs

TABLE III
IEEE 802.11P CHARACTERISTICS FOR DIFFERENT BANDWIDTHS

Parameters	Values
Frequency Range	400Mhz - 6GHz
Channel Bandwidth	5Mhz, 10Mhz, 20MHz
PA granularity	12 steps
IEEE 802.11 PHY	OFDM
Modulation	BPSK, QPSK, 16QAM
IEEE 802.11 MAC	IBSS OCB
Traffic	Broadcast, Unack Unicast
Packet Size	1024 bytes max

TABLE IV
SDR DSRC SPECIFICATIONS

Parameters	Values
Noise Floor	-105 dBm
Noise Figure	11 dB
ED^{th}	-91 dBM

TABLE V
SOFT-MODEL PHY PARAMETERS

1) *Noise Figure* - Without any input from the oscillator, we measured the energy at the receiving end.
2) *Energy Detection Threshold (ED^{th})* - The performance of an IEEE 802.11p chipset depends on the lowest energy to successfully detect and decode successfully an OFDM preamble.
3) *PER vs. SNR* - We computed the packet error rate (PER) curves of the soft-modem for six coding rates (up to 16QAM 2/3).

Results are illustrated in Table V and Fig. 8, and we describe the test procedures below.

1) Noise Figure: The *Noise* in SNR values is composed on channel noise (which we assume here as *Additive White Gaussian - AWGN*) and the *Noise Figure* of the chipset. The latter represents additive noise generated by the electrical components of the chipset and increases the noise level of the RX. The noise figure is measured by subtracting the noise floor (here assumed to be $-105dBm$) to the energy detected at the RX, when no energy is transmitted on the coaxial cable. The value of the noise figure of the Express-MIMO OpenAirInterface chipset has been measured to be $11dB$.

2) Energy Detection Threshold: The Energy Detection threshold (ED_{th}) is a major performance indicator of a soft- or hard-modem, as it indicates the minimum energy at which it can distinguish between noise and signal, and successfully detect a packet at the lowest modulation. The lower it is, the lower can be the RX signal of a packet. ED_{th} also impacts the TX, as the Carrier Sensing threshold (CS_{th}) is usually set to the ED_{th} of the RX. The lower the ED_{th}, the lower the CS_{th} and as such, the TX leaves the RX decode more packets that would otherwise been assumed to be interference.

The ED_{th} measurement is conducted following the process depicted in Fig. 6. Two metrics are measured, which characterize the appropriate value of the ED_{th}:

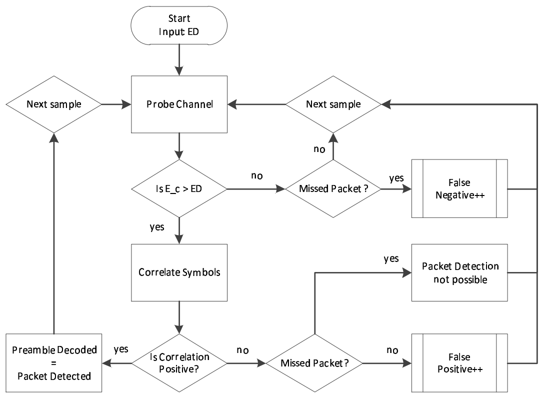


Fig. 6. Energy detection process, giving $\#FalsePositive$ and $\#FalseNegative$ as output.

1) *False Negative* - The RX does not detect any energy on the channel, but a packet was there. This means the ED_{th} is too high and we miss packets.
2) *False Positive* - The RX detects energy on the channel, but it is purely noise or the SNR is too small. This means the ED_{th} is too close to the cumulated $Noise_{floor} + Noise_{figure}$ and is too low to detect any packet.

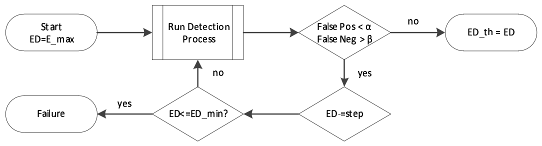
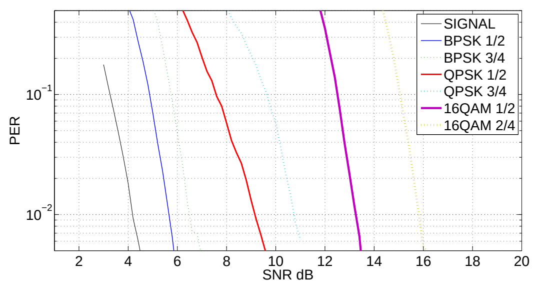


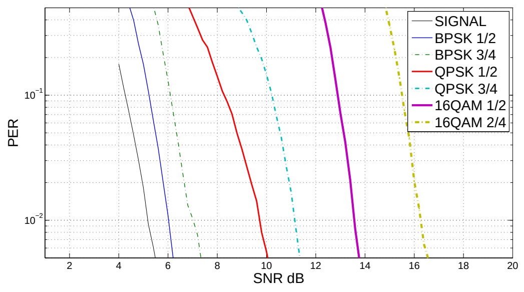
Fig. 7. Energy Detection threshold loop, called the process described in Fig. 6.

To obtain the ED_{th} of the receiver (soft-modem and RF front-end), we ran the algorithm illustrated in Fig. 7. The oscilloscope generates OFDM signals on the channel of various energy levels. The RX periodically (at a rate of $Slottime = 21\mu s$) probes the channel to detect energy above noise. At each iteration, the sum of all false positives and false negatives are measured for the whole training sequence, and as long as the tolerated thresholds are not reached, we reduce the ED value. Upon completion of this test, we obtained a minimum $ED_{th} = -91dBm$. This value is a bit higher than modern DSRC chipsets, but remains below the minimum requirements indicated by [1].

3) PER Curves: Packet Error Rate (PER) curves represent the resilience of the soft- or hard-modem toward noise and interference. The lower the PER, the lower is the required signal to identify symbol in OFDM constellations and decode a packet with high probability. We computed the PER curves of the DSRC soft-modem following the procedure described hereafter. A generated broadcast signal for each modulation schema is iteratively subject to an increasing noise and we measure the ratio of 32-bit CRC mismatch. We evaluated the PER curves for 6 different modulations. Considering the 5Mhz channel bandwidth, the 16QAM 1/2 corresponds to 6Mbps, which is the recommended throughput on the CCH at 5.9GHz.



(a) Packet size 256 Bytes



(b) Packet size 1024 Bytes

Fig. 8. PER vs. SNR curve of the soft-modem for two packet size.

B. *IEEE 802.11p MAC*

Before testing the full prototype, we need to test the extended OCB functionalities of the soft-mac in the linux 802.11 subsystem. The test is to send a 'fake' ping broadcast from the soft-modem up the RX stack, when the driver configured the linux 802.11 subsystem to work in OCB mode. We evaluate first the support of the OCB mode (accepting receptions and transmissions without authentication and associations), and second the transmission delay on the RX stack. The latter is particularly important, considering that the soft-modem communicate over a netlink socket, and as such delay could occur which could become a bottleneck for the crucial end-2-end delay for safety-related transmissions. The soft-modem, driver and 802.11 subsystem modules have been installed in the kernel and the linux iw tool has been used to mount a virtual interface accessible from the Linux IP subsystem.

Setting the driver parameters *dot11OCBActivated=true* on the $wiphy$ structure and setting the flag *DOT11OCB_SUPPORTED* on the hw structure (chipset capabilities), we observed that all Broadcast echo-requests would be delivered to the Linux IP subsystem, and the returned unicast echo-replies would be successfully delivered to the soft-modem. As for the delay, we increased the transmission rate of the 'fake' ping packet and measured the stack delay on the up- and down-stream stack path. Results are depicted in Table VI. We can see that delay remains reasonable considering the soft-modem and driver architecture of the prototype.

VI. Prototyping

In this section, we connected two computers equipped with the SDR IEEE 802.11p prototype. We connected them over a coaxial cable[4]. Our objective is to test the end-to-end connectivity between the Linux 802.11 subsystem over our SDR DSRC prototype. We emulated ITS broadcast and unicast traffic on 802.11 OCB using a Broadcast Ping (echo request in broadcast and echo reply in unicast). On both stations, we mounted the IEEE 802.11p interface as an IBSS interface, configured static IPv4 addresses and disabled ARP[5], and then let STA1 send PING broadcast traffic to STA2 over our IEEE 802.11p prototype. We measured the ping Round Trip time to be stable around $50[ms]$, a value that fits to the relay requirements for safety-related transmissions.

Rate	Throughput	Delay mean	IQR	95% perc.	max
10Hz	0.013 Mbps	69.1019 [μs]	1.5 [μs]	65.5 [μs]	77 [ms]
100Hz	0.13 Mbps	75.3171 [μs]	2 [μs]	70 [μs]	89 [ms]
1000Hz	1.3 Mbps	145.6126 [μs]	2 [μs]	75 [μs]	89 [ms]

TABLE VI
Soft-MAC stack delay, where $rate$ represents the number of packets offered to the stack.

VII. Discussion and Future Work

We presented in this paper OpenAirITS, an open-source software-defined radio DSRC prototype for experimental evaluations of vehicular communication solutions and ITS applications. We described the architecture at different protocol levels and evaluated it in different tests. Notably, we measured key PHY metrics, such as the noise figure, the energy detection threshold, packet error rates and stack delays.

This prototype is built on the OpenAirInterface platform and offers a total access to all IEEE 802.11 MAC and PHY parameters and is also capable of being adapted to different frequency ranges (400Mhz - 6GHz) or channel bandwidth (5MHz - 20 MHz). The DSRC prototype therefore provides a large flexibility for the design and experimental evaluation of the next generation of vehicular communication solutions.

References

[1] IEEE 802.11-2012, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," 2012.
[2] NEC Car2X Communication SDK. [Online]. Available: http://c2x-sdk.neclab.eu/
[3] DENSO Wireless Safety Unit (wsu). [Online]. Available: http://www.denso-europe.com
[4] Cohda Wireless MK2 WAVE-DSRC Radio. [Online]. Available: http://www.cohdawireless.com/product/mk2.html
[5] Drive C2X. [Online]. Available: www.drive-c2x.eu/
[6] T. L. Robert Lasowski and Markus Strassberger, "Openwave Engine / Wsu - A Platform For C2C-CC," in *15th World Congress on Intelligent Transport Systems*, November 2008, pp. 33–40.
[7] S. Biddlestone, "A GNU Radio based testbed implementation with IEEE 1609 WAVE functionality," in *IEEE Vehicular Networking Conference (VNC)*, 2009.
[8] CarGeo6 Project. [Online]. Available: www.cargeo6.org/
[9] P. Fuxjäger *et al.*, "IEEE 802.11p Transmission Using GNURadio," in *6th Karlsruhe Workshop on Software Radios*, 2010.
[10] Proton-plata project. [Online]. Available: http://www.openairinterface.org/node/60

[4]The RF front end has been designed to be connected to omni-directional antenna, as well as car roof antenna

[5]We manually entered the required ARP entries to match the static IPv4 and MAC addresses