Implementation and Performance Evaluation of a New Experimental Platform for Medium Access Control Protocols

Francisco Vázquez Gallego[1], Jesús Alonso-Zarate[1], Danica Gajic[2], Christian Liss[2], and Christos Verikoukis[1]

[1] Centre Tecnològic de Telecomunicacions de Catalunya (CTTC)
Parc Mediterrani de la Tecnologia (PMT), Av. Carl Friedrich Gauss, 7
08860 Castelldefels (Barcelona), Spain
{francisco.vazquez,jesus.alonso,cveri}@cttc.es
[2] InnoRoute GmbH, P.O. Box 260114
80058 Munich, Germany
{gajic,liss}@innoroute.de

Abstract. OpenMAC is presented in this paper as an innovative experimental platform suitable for field testing and performance evaluation of Medium Access Control (MAC) protocols developed in C++. The concept design of OpenMAC avoids the use of hardware-specific code or Hardware Description Language (HDL), softening the learning curve and accelerating the implementation process. This paper describes the OpenMAC hardware/software architecture and shows its benefits with a design example of a Carrier Sense Multiple Access (CSMA) protocol. Finally, the paper provides the implementation details and presents performance results of a practical test to demonstrate how OpenMAC can fulfill strict MAC timing specifications and thus perform as a device backwards compatible with standards.

Keywords: MAC protocol, rapid prototyping, flexibility, cross-layer.

1 Introduction

The Medium Access Control (MAC) layer defines the rules to access a shared communication medium, indicating when and how a specific node is granted access to transmit or receive data. Several MAC protocols have been proposed in the literature due to their key impact into the performance and energy-efficiency of communication networks. However, most of these works present only theoretical analysis or simulation, and just in very few cases they have been actually tested in the field. Experimental prototyping and real testing are essential processes for evaluating the protocols under realistic physical (PHY) layer conditions due to the fact that the inclusion of real-world effects in theoretical models often leads to intractable problems and computer-based simulation techniques are affected by the variability and, usually lack PHY layer accuracy [1].

The requirements for a MAC experimental platform are tightly related to the key functions of the MAC layer, which are briefly described as follows:

1) Provide access to PHY layer information, e.g., to monitor the channel status for Clear Channel Assessment (CCA) in order to know whether the channel is idle or busy.
2) Enable PHY layer re-configuration to select various PHY parameters such as data-rate, modulation, transmit-power, frequency-channel, etc.
3) Enable precise scheduling and timing, e.g., to ensure that transmissions occur during time-slots in Time Division Multiple Access (TDMA) protocols.
4) Guarantee fast identification of received control packets which have to be quickly processed to generate answers and make decisions rapidly.
5) Guarantee fast packet transmission, e.g., in contention-based protocols packets must be transmitted with minimum delay when the channel is detected to be clear.

In addition, an ideal platform should also:

1) Smooth the learning curve to create a prototype from a protocol concept design.
2) Guarantee the fulfillment of the protocol time-specifications to ensure backwards compatibility with standards in case it is required.
3) Provide flexible interfacing to experiment with different PHY layers.

A number of research projects have addressed the implementation of experimental platforms to test MAC protocols [2]. Two categories of platforms have been identified in the literature: platforms based on commercial Network Interface Cards (NIC) and platforms based on custom hardware.

Commercial off-the-shelf 802.11 NIC-based platforms [3]-[5] split the implementation of the MAC layer functions between the firmware on the networking card, which implements the time-critical functions, and the software-driver running on the computer hosting the card, which implements the time-tolerant functions. NIC-based solutions constitute an inexpensive prototyping approach and facilitate reprogramming the MAC protocol functions by modifying the software-driver. However, the interface between the host-computer and the NIC introduces unpredictable delays which may compromise the precise scheduling and the compliance with strict timing requirements of the MAC protocol. For this reason, time-critical functions are programmed in the NIC firmware, which usually cannot be easily reconfigured, thus limiting the flexibility to efficiently customize certain MAC functions. This limitation is extended to the access to information and configuration of the PHY layer, thus not allowing for MAC-PHY cross-layer designs.

Several custom hardware platforms [6][7] have been implemented as an alternative to NIC-based platforms, e.g., TUTWLAN [8], Universal Software Radio Peripheral (USRP) [9] and GNU Radio [10], Wireless Open-Access Research Platform (WARP) [11], CalRadio [12], OpenAirInterface [13], among others. Custom hardware platforms are equipped with an analog Radio-Frequency (RF) front-end connected to a built-in computing subsystem by means of analog-to-digital and digital-to-analog converters. The computing subsystem performs both the PHY digital base-band processing (BBP) and the MAC functions. These solutions are based on Field-Programmable-Gate-Arrays (FPGAs), embedded processors, or Digital-Signal-Processors (DSP). They offer full control over the MAC and the PHY layers. The implementations based on FPGA allow partitioning the MAC into software and custom hardware-accelerators, offering even more processing power. In addition, they enable to experiment with different PHY layers and to design MAC-PHY cross-layer optimizations.

On the other hand, state of the art custom-hardware platforms have some disadvantages. Their implementation cost is very high and they show a steep learning curve to start prototyping. The protocol designer needs to understand the hardware details to develop hardware-specific C-code in order to fulfill the tightest protocol time-constraints. However, MAC designers are typically familiar with high-level programming languages, e.g., C++, which may endanger the implementation of time-critical functions. While the use of FPGAs allows hardware-acceleration and may relax the software optimization requirements, it forces the protocol designer to learn a Hardware Description Language (HDL). This is the main motivation for the development of the OpenMAC platform, which intends to bring MAC prototyping capabilities closer to the research community. OpenMAC introduces a special hardware/software partitioning of the MAC protocol that eases the task to fulfill the most challenging MAC timing-specifications with non hardware-specific code. The protocols can be entirely designed in C++ and hardware is transparent to the designer.

The contribution of this paper is threefold. Firstly, it describes the implementation of the OpenMAC platform architecture presented in [14], which is summarized in this paper to make it self-contained. Secondly, it provides a comprehensive description of the measurement procedure and performance evaluation of the platform. Thirdly, it provides an example of implementation of a contention-based protocol.

The remainder of this paper is organized as follows. Section 2 briefly describes the hardware/software architecture of OpenMAC. Section 3 provides a code example for a MAC protocol. The hardware implementation of OpenMAC is described in Section 4. The performance results of the OpenMAC platform are reported and discussed in Section 5. Finally, Section 6 concludes the paper.

2 OpenMAC Platform Architecture

The architecture of the OpenMAC platform is depicted in Fig. 1. The implementation of a MAC protocol is split into the Software (SW) MAC and the Hardware (HW) MAC modules. The reason for this HW/SW partitioning is to enable the use of *non-hardware-specific C/C++ code* for protocol prototyping and to *fulfill the MAC protocol time specifications.*

The SW MAC implements non time-critical MAC functions in C++ code. The SW MAC is prevented from accessing long data payloads in order to minimize non-deterministic software latencies. Instead, the SW MAC only needs to access the packet headers and packet-descriptors to make simple MAC decisions.

The HW MAC executes time-critical MAC operations and heavy computation functions by means of hardware acceleration, e.g., Clear Channel Assessment (CCA), packet error control, encryption, etc. In addition, the HW MAC provides an efficient hardware mechanism to guarantee precise and deterministic MAC scheduling, e.g., time slots in reservation-based protocols, and back-off periods and inter-frame spacing in contention-based protocols.

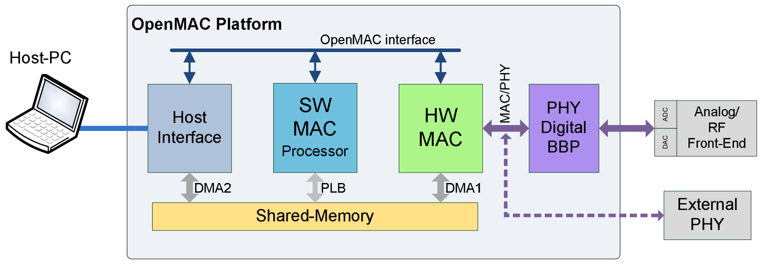

Fig. 1. Hardware/Software architecture of the OpenMAC platform

The HW MAC interfaces to the PHY digital BBP through the MAC/PHY interface, which supports fast transfer of packets and control parameters. The HW MAC configures various PHY parameters on the BBP such as the data-rate, modulation, transmit power, and carrier selection. In a similar way, the HW MAC monitors some variables that are read from the PHY layer, e.g., Received Signal Strength Indicator (RSSI). Access to PHY layer information at the MAC and upper layers is important in order to support cross-layer optimizations.

The layers above the MAC are integrated in a host computer (Host-PC) connected to the OpenMAC platform through the Host Interface. However, the upper layers can also be integrated in the SW MAC processor when stand-alone operation is required. The current design of the Host Interface has the same architecture as the HW MAC.

The shared-memory is a common storage place for accelerating the packet transactions among the SW MAC, HW MAC, and the Host-PC by means of a Direct Memory Access (DMA). The SW MAC accesses the shared-memory through the Processor Local Bus (PLB). The HW MAC stores in the shared-memory the packets received from the PHY, and it reads the ones to be transmitted through the PHY. Similarly, the Host-PC stores in the shared-memory the packets coming from upper layers and reads the ones stored by the HW MAC in the reception process.

Each packet is referenced by a packet-pointer that identifies the base address where the packet is located in the shared-memory. The SW MAC exchanges packet-pointers with the HW MAC and with the Host-PC through the OpenMAC interface.

The remainder of this section is organized as follows. Section 2.1 describes the structure of the packets in the shared-memory. The operation of the HW MAC blocks is detailed in Section 2.2. Finally, Section 2.3 describes the SW MAC functionalities and details the functions of the Hardware Abstraction Layer of OpenMAC.

2.1 Packet-Descriptors

A packet in the shared-memory is comprised of three components:

1) The packet-header contains control information related to the MAC protocol.
2) The data-payload contains the actual useful data of the packet.
3) The packet-descriptors contain control and status information required for transmission and reception.

Two types of packet-descriptors are considered: MAC and PHY descriptors. The structure and contents of these packet descriptors are summarized in Table 1 and Table 2 for the transmission and reception processes, respectively.

Table 1. Packet-descriptors used in packet transmission

MAC Layer descriptors	
PacketLength	Length of packet-header and data-payload in bytes
TimeStamp	Time when the SW MAC wants to start transmitting a packet
EnableRxTimeOut	To enable/disable the HW MAC to detect a time-out event in packet-reception after a packet is transmitted
EnableBackoff	To enable/disable the HW MAC to perform backoff process
RxTimeOut	Time-out interval to be used by the HW MAC to receive a new packet just after a packet-transmission if EnableRxTimeOut is enabled
Backoff	Max. duration of the backoff period to be used by the HW MAC for the transmission of a packet when EnableBackoff is enabled
PHY Layer descriptors	
Modulation	Modulation type used for a transmission
DataRate	Data-rate used for transmission at the PHY layer
TxPower	Transmission power
Carrier	Frequency band and carriers used for transmission

Table 2. Packet-descriptors used in packet reception

MAC Layer descriptors	
PacketLength	Length of packet-header and data-payload in bytes
TimeStamp	Time when the packet was received from the PHY layer
CRCresult	Indicates if the current packet was received without error
PHY Layer descriptors	
Modulation	Modulation type used for the reception of a packet
DataRate	Data rate at which the packet was received at the PHY layer
RSSI	The RSSI is a measure of the RF energy received

The SW MAC can read or write the packet-descriptors by accessing directly to the packets' structures in the shared-memory. The HW MAC transfers the PHY layer packet-descriptors among the shared-memory and the PHY layer.

2.2 Hardware MAC

The HW MAC is a digital module designed in VHDL. It has been created to support any time-critical function that can be required by any type of MAC protocols (i.e., contention-based, reservation-based, and hybrid MAC protocols), and to avoid re-design efforts for adapting the HW MAC to new functions.

In this section, the functionalities and the interaction among the HW MAC elements are described. A block diagram of the HW MAC is shown in Fig. 2.

The connection between the HW MAC and the SW MAC modules is implemented with the two interfaces shown in the left hand side of the diagram: the DMA bus to the shared-memory and the OpenMAC interface directly to the SW MAC.

The transfer of packets is performed through the shared-memory using the DMA bus. In the transmission process, packets are read from the shared-memory by the Tx DMA block and then parsed to extract the packet-descriptors at the Tx Packet Parsing block. In the reception process, the packet-descriptors are first collected and registered by the Rx Packet Formatting block, and then the packets are written into the shared-memory by the Rx DMA block.

The SW MAC exchanges packet-pointers (PP) with the Tx PP Manager and Rx PP Manager blocks through the OpenMAC interface. The packet-pointers are temporarily buffered in the TxPP FIFO and RxPP FIFO blocks, respectively, for transmission and reception. In the transmission process, the SW MAC sends the packet-pointers to the Tx PP Manager block. In the reception process, the Rx PP Manager notifies to the SW MAC when a new packet has been stored in the shared-memory. In the case that the RxPP FIFO is empty, the Rx PP Manager block notifies to the SW MAC, which sends one or more available packet-pointers. This approach overcomes possible latencies introduced by the SW MAC and guarantees that packets received from the PHY layer can be stored into the shared-memory immediately.

As shown in Fig. 2, the OpenMAC interface is connected to the internal registers of the Channel State Monitor and the Configuration BBP/RF blocks. In their turn, these two are connected to the PHY layer through the MAC/PHY interface, shown in the right hand side of the diagram. This interface supports hardware-accelerated transfers of data and control parameters between the HW MAC and the PHY layer. The Channel State Monitor block periodically senses the channel status (e.g., RSSI, busy, or idle) provided by the PHY layer. The channel status is read by the Backoff Controller block, or by the SW MAC, to implement CSMA.

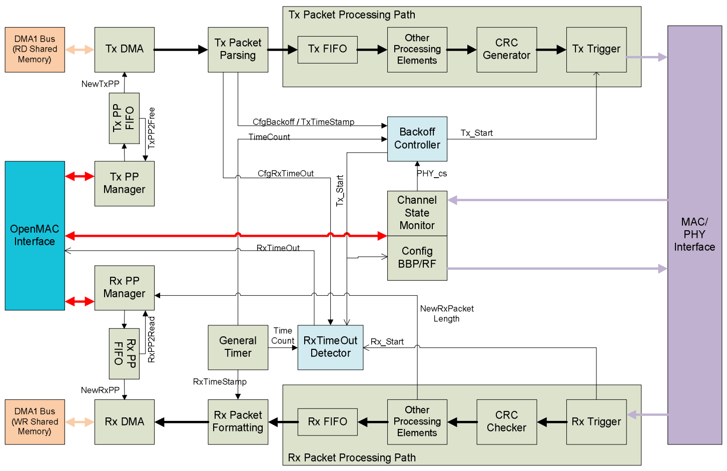

Fig. 2. Hardware MAC Module Functional Block Diagram

The Configuration BBP/RF block configures the PHY layer (i.e., the Digital BBP and the Analog RF Front-End) by transferring the contents of its configuration registers (e.g., modulation, transmit power, carrier selection, etc). The configuration registers can be updated at any time from the SW MAC through the OpenMAC interface. In addition, the PHY configuration can be performed in a packet by packet basis by sending the PHY packet-descriptors (detailed in Table 1) at the beginning of each packet transmission.

The HW MAC includes two packet processing paths devoted to the execution of time-critical functions in packet transmission and reception, e.g., Cyclic Redundancy Code (CRC), encryption, packet aggregation, etc. Packets are fetched from the shared memory, processed, and transferred to the MAC/PHY interface, and inversely with packets received from PHY.

Finally, the HW MAC includes three hardware blocks devoted to provide simple, precise and deterministic scheduling and timing to the MAC layer:

1) The General Timer block provides a time indication signal (TimeCount) to the other blocks with a time resolution of 1 μs.
2) The RxTimeOut Detector block detects when no packet is received within a time period (i.e., RxTimeOut packet-descriptor in Table 1) after the last transmission, e.g., acknowledgement (ACK) is lost for the previous data-packet transmission.
3) The Backoff Controller block determines the time instants to trigger the packet transmission by comparing the TimeCount signal with the TimeStamp packet-descriptor. It can be configured to execute a backoff process using the Backoff packet-descriptors, detailed in Table 1, and the channel status (PHY_cs) provided by the Channel State Monitor block.

2.3 Software MAC

The SW MAC is a C++ program that runs on an embedded processor without operating system. While the packet-processing load is concentrated on the HW MAC, which transfers great volumes of data through the shared-memory to exchange packets with the Host-PC, the SW MAC only accesses the packet-descriptors and packet-headers to make simple MAC protocol decisions (e.g., check the destination address, identify the packet type, determine time-stamps for transmission, etc). Therefore, by avoiding the need to access long data-payloads from software, the OpenMAC platform reduces software latencies and thus facilitates the fulfillment of strict timing constraints of the MAC protocol without using hardware specific code.

The use of packet-descriptors (described in Section 2.1) simplifies the access from the SW MAC to the hardware, i.e., HW MAC, Host Interface, and PHY layer parameters. The SW MAC simply performs fast read or write access to the packet-descriptor stored in the shared-memory that needs to monitor or configure. This approach avoids the use of dedicated and complex software functions to access the PHY, reduces the complexity in the development of the MAC protocol, and minimizes delays.

In order to make the hardware details transparent to the protocol designer, a Hardware Abstraction Layer (HAL) is included in OpenMAC. The HAL is comprised of a set of software functions which ease the access to the HW MAC and the Host interface. A HAL function is a very short piece of C-code that executes read/write accesses to the hardware internal registers through the OpenMAC interface.

The functions of the OpenMAC HAL are described in Table 3. The input argument named *channel* used in all the HAL functions allows selecting among the channels connected to the OpenMAC interface bus: the PHY channel (HW MAC) and the HOST channel (Host interface). Both channels have basically the same architecture as the HW MAC module described in Section 2.2.

Table 3. Functions of the OpenMAC Hardware Abstraction Layer

Function: *`packet_pointer get_packet(char channel)`*
Description: It returns the packet-pointer of a new packet that has been received from *channel* and has been stored in the shared-memory.
Function: *`void send_packet(packet_pointer  tx_pointer, char channel, unsigned int timestamp)`*
Description: It writes the packet-pointer *tx_pointer* of the next packet to transmit through the *channel*. The transmission will start at the time indicated by the packet-descriptor *timestamp*, passed as an argument.
Function: *`unsigned int check_send_packet(char channel)`*
Description: It returns the integer number of packets that are still pending to be transmitted through the *channel*. The packets are buffered in the Tx FIFO of the *channel*.
Function: `unsigned int check_get_packet(char channel)`
Description: It returns an integer value that can be mapped into the state of the reception process: (1) a new packet has been received from the *channel* and stored in the shared-memory, (2) the *channel* needs packet-pointers for the reception of new packets, and (3) a time-out event has been detected after the last transmitted packet.
Function: *`void pointer_allocation(char channel, char n, packet_pointer *rx_pointer)`*
Description: It writes into the RxPP FIFO of the *channel* a list of *n* packet-pointers, *rx_pointer*, which will be used to store in the shared-memory new packets received.
Function: *`unsigned int get_phy_cs(char channel)`*
Description: It returns the status (RSSI, busy, idle) of the PHY layer connected to channel.
Function: *`void config_phy(char channel, unsigned short modulation, unsigned short data_rate, unsigned short tx_power, unsigned short carrier_select)`*
Description: It writes the parameters, passed as input arguments, to configure the channel.
Function: *`unsigned int get_timer(char channel)`*
Description: It returns the current TimeCount value of the General Timer shown in Fig. 2.
Function: *`void reset_timer(char channel)`*
Description: It resets the General Timer block of the *channel*.

3 Example of CSMA Protocol

This section presents an example of implementation of a CSMA protocol in the SW MAC processor. The aim of this description is to demonstrate how simple the design of a MAC protocol using OpenMAC can be.

In CSMA, a node that has data to transmit first senses the channel. In the case that the channel is clear, the node starts transmitting. Otherwise, the node initiates a random backoff and waits for the channel to be idle. The backoff timer is decremented along time; it is frozen when the channel is busy, and it is resumed when the channel becomes idle again. When the backoff timer expires, i.e., gets to zero, the packet is transmitted if the channel is idle, otherwise the backoff is reinitiated. When a data-packet is sent, the transmitting node waits for an ACK from the intended destination. If the ACK is not received within a timeout, the packet is retransmitted.

The SW MAC code for CSMA is shown below. It is divided in two parts:

(1) Transmission of Data-Packets from Host-PC to PHY. The data-packets transmitted from the Host-PC to the PHY are temporarily stored in the variable *buffer_HOST2PHY* of type packet. A new packet can only be stored in the *buffer_HOST2PHY* when it is free, i.e., an ACK has been received for the previously transmitted data-packet. The hardware Backoff Controller implements the backoff process and the RxTimeOut Detector generates the ACK time-out events. Therefore, the SW MAC has only to configure the associated packet-descriptors (i.e., EnableBackoff, Backoff, EnableRxTimeOut, and RxTimeOut) on each data-packet transmission. If an ACK time-out event is detected, the data-packet is re-transmitted.

(2) Reception of Packets from PHY to Host-PC. The packets received from PHY are temporarily stored in the *buffer_PHY2HOST*, of type packet, to be later transferred to the Host-PC. Since the throughput among the Host-PC and OpenMAC can be assumed higher than the throughput among OpenMAC and PHY, the *buffer_PHY2HOST* is always ready to store new received packets. The SW MAC first checks the Cyclic Redundancy Code (CRC) descriptor of the received packet. Then, the SW MAC checks the destination address and the packet type. If it is a data-packet, it is transferred to the Host-PC and the SW MAC starts the transmission of an ACK to the PHY delayed a Short Inter Frame Space (SIFS) from the time-stamp of the received packet. If an ACK is received, the *buffer_HOST2PHY* is released.

Example code of a CSMA protocol implemented in the SW MAC processor.

```
packet_pointer TxPP, RxPP;
packet buffer_HOST2PHY, buffer_PHY2HOST, ACK;
unsigned int status_HOST = 0, status_PHY = 0; char buffer_ready = 1;
while(1){
/* PART 1: transmission of data-packets from Host-PC to PHY */
  status_HOST = check_get_packet(HOST);
  /* Packet-pointer requested for data-packet reception from HOST? */
  if ((status_HOST & RxPP_REQUEST) && buffer_ready)
       pointer_allocation(HOST, 1, &buffer_HOST2PHY);
  /* Data-packet stored in buffer_HOST2PHY by the HOST_IF? */
  if (status_HOST & RxPACKET_RDY) {
       buffer_ready = 0; /* buffer_HOST2PHY is busy with new packet */
       TxPP = get_packet(HOST); /* get the packet-pointer */
       /* write some of the MAC and PHY packet-descriptors */
       TxPP-> EnableRxTimeOut = ENABLED; /* enable Rx time-out */
       TxPP-> RxTimeOut = ACKTIMEOUT; /* time-out to receive ACK */
       TxPP-> EnableBackoff = ENABLED; /* enable backoff */
       TxPP-> Backoff = CW; /* Contention-Window for backoff process */
```

```
        TxPP-> TxPower = TX10dBm; /* configure PHY tx power */
        send_packet(TxPP, PHY, 0); /* transfer the packet to HW MAC */
        }
   /* Time-out detected after the last data-packet sent to PHY? */
   if (status_PHY & RxTIMEOUT) send_packet(TxPP, PHY, 0);/* tx again*/

/* PART 2: reception of packets from PHY to Host-PC */
   status_PHY = check_get_packet(PHY);
   /* Packet-pointer requested for packet reception from PHY? */
   if (status_PHY & RxPP_request)
        pointer_allocation(PHY, 1, &buffer_PHY2HOST);
   /* Packet stored in buffer_PHY2HOST by the HW MAC? */
   if (status_PHY & RxPACKET_RDY){
        RxPP = get_packet(PHY); /* get the packet-pointer */
        if (RxPP->CRCresult == OK){ /* check that CRC is correct */
           for (i=0; i<6; ) /* check the destination MAC address */
                    if (RxPP->DstAddr[i] == my_Addr[i]) i++; else break;
           if (i==6){ /* destination MAC address is correct? */
                 send_packet(RxPP, HOST, 0); /* transfer to HOST */
                 if (RxPP->Type == DATA){ /* is it a data-packet? */
                    /* calculate time-stamp to transmit an ACK */
                    TxTimeStamp = RxPP->TimeStamp + SIFS;
                    /* transfer ACK to HW MAC */
                    send_packet(&ACK, PHY, TxTimeStamp);
                    } /* is it an ACK? then release the buffer_HOST2PHY */
                 else if (RxPP->Type == ACK) buffer_ready = 1;
                 }
        }
    }
}
```

4 Hardware Implementation

The OpenMAC platform architecture has been implemented and tested on a Commercial Off-The-Shelf (COTS) FPGA-based development board. The technical features of this board are detailed in Table 4. It contains a Xilinx Virtex-5 FPGA [15] which integrates two PowerPC440 processors. One of the processors is used to run the code of the SW MAC and the other is left unused for future applications, e.g., development of high-performance upper layers for standalone OpenMAC nodes. The rest of the FPGA implements the following digital modules: the HW MAC, the Host Interface, the OpenMAC interface bus, the shared-memory, and the MAC/PHY interface.

Table 4. Features of the FPGA-based Development Board

Board part-number	HTG-V5-PCIE-100-2 (from HiTech Global, LLC [16])
FPGA	Xilinx Virtex-5 XC5VFX100T-3
Processors	2 PowerPC 440 (hard cores)
RAM Memory	512MB DDR2-SDRAM, expandable up to 2GB
Flash Memory	4 MB
Interface with Host	8-lanes PCI Express End-Point connector. 2 Gigabit Ethernet ports.
Interface with PHY	2 connectors (QSE type) with 64 pairs of Low Voltage Differential Signals (LVDS) or 128 single-ended signals. Outputs of 5V and 3.3V supply for add-on PHY modules.

The shared-memory module is implemented using on-chip Random Access Memory (RAM) blocks of the FPGA. The Double Data Rate-Synchronous Dynamic RAM (DDR-SDRAM) module, included in the FPGA-based board, is left unused for future extensions with huge memory requirements.

The PowerPC440 processor clock has been set to 400MHz in order to provide high speed to the SW MAC. For the processor peripheral cores (i.e., HW MAC, Host Interface, and shared-memory) and for the communication buses (i.e., OpenMAC interface, PLB, and DMA buses) the clock frequency has been selected as low as 100MHz in order to facilitate and speed-up the FPGA implementation process.

The FPGA-based board includes one Peripheral Component Interconnect (PCI) bus and two Gigabit Ethernet PHY ports based on the Alaska 88E1111 transceiver from Marvell. One of the Ethernet ports is used to connect the OpenMAC platform with the Host-PC. The PCI bus is foreseen to plug the board into a desktop computer in applications with greater bandwidth and throughput requirements.

The expansion connectors of the FPGA-based board allow the implementation of a flexible MAC/PHY interface with external PHY layer boards. Two different PHY layer solutions can be implemented:

1) A complete PHY which incorporates the BBP and the Analog RF Front-End.
2) The BBP can be implemented inside the FPGA and an external Analog RF Front-End can be used for the RF operation.

So far, a Gigabit Ethernet PHY has been used for the proof-of-concept of OpenMAC. The use of a wired solution is mainly motivated by the lack of availability on the market of low-cost high-performance wireless PHY solutions which do not require great implementation efforts.

5 Performance Evaluation

The aim of this section is to show the ability of the OpenMAC platform to meet the time-constraints of a standardized MAC protocol. As an example, in the context of the IEEE 802.11 Standard for Wireless Local Area Networks [17], the data-packet reception and post processing to decide whether to transmit an ACK or not, and the initialization of the transmission of the ACK, must be done within less than a SIFS interval, i.e., tens of microseconds, in order to be able to tolerate non-negligible propagation delays as well.

The remainder of this section is organized as follows. First, Section 5.1 details the performance metrics. The experimental setup is briefly described in Section 5.2 and, finally, the performance results are discussed in Section 5.3.

5.1 Performance Metrics

The following metrics have been considered for the performance evaluation of OpenMAC. The time metrics are shown in the timing diagram of Fig. 3.

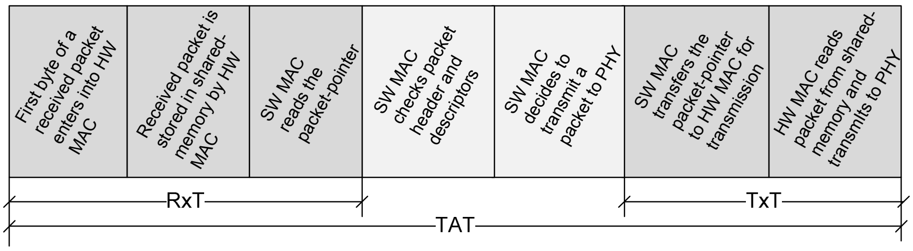
First byte of a received packet enters into HW MAC
Received packet is stored in shared-memory by HW MAC
SW MAC reads the packet-pointer
SW MAC checks packet header and descriptors
SW MAC decides to transmit a packet to PHY
SW MAC transfers the packet-pointer to HW MAC for transmission
HW MAC reads packet from shared-memory and transmits to PHY
RxT
TxT
TAT

Fig. 3. Timing diagram of the HW MAC and SW MAC processes within TxT, RxT, and TAT

- **Transmission Time (TxT)** is defined as the time elapsed from the moment when the SW MAC initiates the transfer of the associated packet-pointer to the HW MAC, until the moment when the HW MAC finishes the transmission of the packet to the PHY.
- **Throughput in transmission** is defined as the length of transmitted packets divided by the corresponding TxT (assuming a constant packet length).
- **Reception Time (RxT)** is defined as the time elapsed from the moment when the first byte of a received packet enters into the HW MAC, until the moment when the SW MAC reads the packet-pointer.
- **Throughput in reception** is defined as the length of the received packets divided by the corresponding RxT (assuming a constant packet length).
- **Turn-Around Time (TAT)** is defined as the time elapsed from the moment when the first byte of a received packet enters into the HW MAC, until the moment when the HW MAC finishes the transmission of an answer packet (ACK) to the PHY. During this time, the SW MAC reads the packet-pointer, checks the packet header and descriptors, and transfers the ACK packet-pointer.

It is worth noting that the processing delays introduced by the PHY layer are not considered in the definition of TxT, RxT, and TAT. The TAT is the most critical time parameter for those protocols which rely on handshaking of control packets, e.g., the ACKs used in the IEEE 802.11 Standard, as it defines the capability to respond to received packets in due time. This parameter is the key to ensure backwards compatibility with current standards.

5.2 Experimental Setup

The performance of the OpenMAC platform architecture has been assessed using the board described in Section 4. The wired Gigabit Ethernet PHY layer has been used for the setup shown in Fig. 4, which does not include the Host Interface for simplicity. One of the Ethernet transceivers has been connected through its Gigabit Media Independent Interface (GMII) to the MAC/PHY interface in the FPGA. Then, the OpenMAC platform has been connected to a Gigabit Ethernet port of a computer.

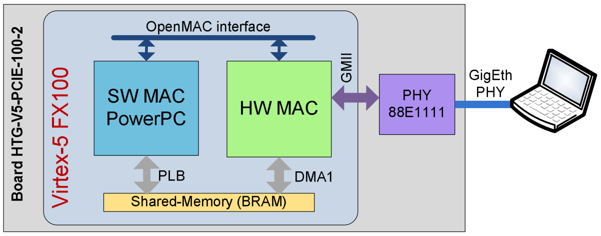

Fig. 4. Experimental setup used for the performance tests of OpenMAC

The following procedure has been used to measure the TxT, RxT, TAT, and the different throughput values.

1) The OpenMAC platform transmits or receives packets to/from the computer through the Gigabit Ethernet PHY.
2) The following HAL functions, described in Section 2.3, are included in a test program to receive, to transmit, to monitor the status, and to allocate packet-pointers:
   get_packet(), *send_packet()*, *check_get_packet()*, *check_send_packet()*, and *pointer_allocation()*.
3) The *get_timer()* function is utilized to measure time periods between two different instants. In this way, the time measurements are simplified by employing the time-base provided by the HW MAC.
4) In order to transmit Ethernet packets with a controlled length to the OpenMAC platform, the *ping* command is executed on the desktop computer as:
   ping -t -l length ip_address_openMAC .
5) The transmission and reception throughput values have been computed by dividing the packet length by the TxT and RxT measurements, respectively.

5.3 Experimental Results

The experiments have been performed with different data-packet lengths (100, 500, 1000, and 1500 bytes) and ACK packets of 46 bytes, which is the minimum data field length in Ethernet frames. Fig. 5 shows the results for TxT, throughput in transmission, RxT, throughput in reception, and TAT.

In terms of throughput, OpenMAC performs close to the maximum available throughput in Gigabit Ethernet with long data-packets (1000 to 1500 bytes), which is 1Gbps. As far as the turn-around-time values are concerned, OpenMAC achieves values of the TAT that are below 16 µs for short packets (46 to 500 bytes) and below 20 µs for long packets (1000 to 1500 bytes). It is worth noting that, for short packets, and if the PHY layer device connected to OpenMAC was changed to one compliant with the IEEE 802.11 Standard [17], OpenMAC would fulfill the SIFS time (16 µs) specified in the standard for OFDM PHY layer using a bandwidth of 20MHz. Similarly, for long packets, OpenMAC is in compliance with the SIFS time (32 µs) using a bandwidth of 10MHz. In addition, the TAT could still be reduced by increasing the clock frequency of the processor peripheral cores (i.e., HW MAC, and shared-memory) and of the communication buses (i.e., OpenMAC interface, PLB, and DMA buses) above the currently utilized 100MHz. This frequency adjustment could yield a reduction of more than 20% of the TxT, RxT, and TAT results obtained in the current implementation.

The measurements that have been carried out on the FPGA board demonstrate that the OpenMAC platform fulfils the strictest timing specifications of the IEEE 802.11 Standard.

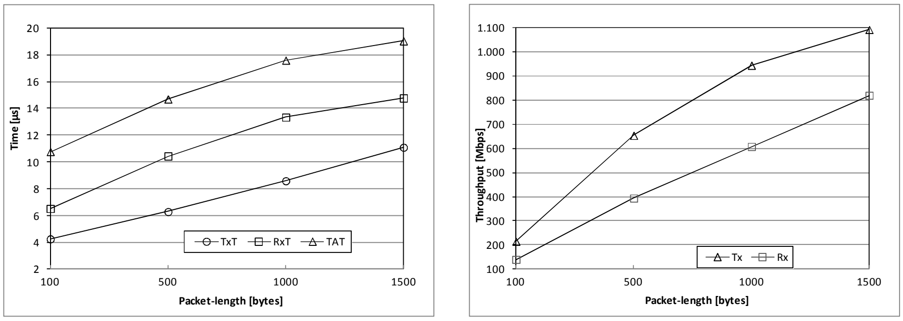

Fig. 5. (a) TxT, RxT, TAT, and (b) throughput in Tx/Rx (with respect to the packet-length)

6 Conclusions

OpenMAC has been presented in this paper as an innovative platform suitable for the development and experimental testing of MAC protocols. It has been implemented using a COTS FPGA-based board with a processor integrated in the FPGA. OpenMAC provides full flexibility and re-configurability to develop any MAC protocol, to change the PHY layer, and to do research with cross-layer optimizations. The main objectives considered in the design of OpenMAC have been:

1) To speed-up the development process of a MAC protocol by simplifying the conversion from the theoretical model down to implementation,
2) To fulfill the protocol time-constraints, and
3) To attain precise scheduling.

Regarding the complexity of the protocol development cycle, OpenMAC allows implementing MAC protocols entirely designed in C++, without operating system in the embedded processor, and with hardware fully transparent to the protocol designer, who does not have to optimize hardware-specific C, assembly or HDL code.

OpenMAC integrates a set of software HAL functions that abstract the accesses to hardware. In addition, it delegates the packet processing to hardware accelerators and prevents software from accessing high-volume data-payloads. The software accesses only short packet-descriptors and packet-headers, which facilitate the fulfillment of MAC time constraints. Moreover, OpenMAC contains hardware accelerators devoted to configure and monitor the PHY layer, and to perform backoff processes, which avoids software to fast access to PHY. Regarding scheduling and timing, OpenMAC includes hardware elements to accurately control time events.

The performance of the OpenMAC platform has been tested experimentally using Gigabit Ethernet PHY. It performs at maximum throughput of 1 Gbps and yields turn-around-times below 20 μs for long packets (1000 to 1500 bytes) and below 16 μs for short packets (46 to 500 bytes). Hence, OpenMAC can fulfill strict MAC timing specifications such as the ones of the IEEE 802.11 Standard.

OpenMAC achieves successfully the aforementioned objectives and brings prototyping capabilities closer to the research community by overcoming the limitations of state of the art. Future work will be aimed at attaching a custom wireless PHY layer (e.g., the WARP reference design [11]) to the OpenMAC platform. Furthermore, the execution of the OpenMAC software will be accelerated by using hardware interrupts, instead of software polling, to notify the state of the OpenMAC hardware modules.

Acknowledgements. This work is supported by the Research Projects CO2GREEN (TEC2010-20823), GREEN-T (TSI-020400-2011-16), and GREENET (PITN–GA–2010-264759).

References

1. Kotz, D., Newport, C., Gray, R.S., Liu, J., Yuan, Y., Elliott, C.: Experimental evaluation of wireless simulation assumptions. In: Proceedings of MSWiM (2004)
2. Vázquez Gallego, F., Alonso-Zarate, J., Alonso, L., Verikoukis, C.: A Survey on Prototyping Platforms for the Development and Experimental Evaluation of Medium Access Control Protocols. IEEE Wireless Communication Magazine 19(1), 74–81 (2012)
3. Korakis, T., Knox, M., Erkip, E., Panwar, S.: Cooperative Network Implementation Using Open-Source Platforms. IEEE Communications Magazine 47(2), 134–141 (2009)
4. Lu, M.H., Steenkiste, P., Chen, T.: Using Commodity Hardware Platform to Develop and Evaluate CSMA Protocols. In: Proceedings of the Third ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization, San Francisco, pp. 73–80 (2008)
5. Verikoukis, C., Pérez-Neira, A., Alonso-Zárate, J., Skianis, C.: Experimental Performance Evaluation of a MAC Protocol for Cooperative ARQ Scenarios. In: Proc. of the IEEE GLOBECOM, Hawaii (2009)
6. Pawelczak, P., Nolan, K., Doyle, L., Oh, S.W., Cabric, D.: Cognitive radio: Ten years of experimentation and development. IEEE Communications Magazine 49(3), 90–100 (2011)
7. Chowdhury, K.R., Melodia, T.: Platforms and Testbeds for Experimental Evaluation of Cognitive Ad Hoc Networks. IEEE Communications Magazine 48(9), 96–104 (2010)
8. Tikkanen, K., Hännikäinen, M., Hämäläinen, T., Saarinen, J.: Advanced Prototype Platform for a Wireless Multimedia Local Area Network. In: 10th European signal processing conference (EUSIPCO), Tampere, pp. 2309–2312 (2000)
9. Nychis, G., Hottelier, T., Yang, Z., Seshan, S., Steenkiste, P.: Enabling MAC Protocol Implementations on Software-Defined Radios. In: Proceedings of the USENIX NSDI, Boston (2009)

10. Blossom, E.: Exploring GNU Radio (2004), `http://www.gnu.org/software/gnuradio/doc/exploring-gnuradio.html`
11. Hunter, C., Camp, J., Murphy, P., Sabharwal, A., Dick, C.: A flexible framework for wireless medium access protocols. In: Asilomar (2006)
12. Manfrin, R., Zanella, A., Zorzi, M.: Functional and Performance Analysis of CalRadio 1 platform. In: 8th IEEE International Symposium on Network Computing and Applications, Cambridge, pp. 300–305 (2009)
13. OpenAirInterface, `http://www.openairinterface.org/`
14. Vázquez Gallego, F., Alonso-Zarate, J., Liss, C., Verikoukis, C.: OpenMAC: A New Reconfigurable Experimental Platform for Energy-Efficient Medium Access Control Protocols. IET Science, Measurement & Technology Journal (in press)
15. Xilinx, Virtex-5 FXT FPGAs, `http://www.xilinx.com/products/virtex5/fxt.html`
16. HiTech Global LLC, `http://www.hitechglobal.com/`
17. IEEE Std. 802.11-2007, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications (2007)