

	<p style="text-align: center;">MOBILEMAN</p> <p style="text-align: center;">IST-2001-38113</p> <p style="text-align: center;">Mobile Metropolitan Ad hoc Networks</p>
---	--

<p>MOBILEMAN</p> <p>MobileMAN network card</p> <p>Deliverable D12</p> <p>Contractual Preparation Date: January 2005 Actual Date of Delivery: 25 February 2005 Estimated Person Months: 37 Number of pages: 27</p> <p>Contributing Partners: Scuola Universitaria Professionale della Svizzera Italiana (Switzerland)</p> <p>Authors: Ralph Bernasconi, Ivan Defilippis, Silvia Giordano, Alessandro Puiatti</p>
--

<p>Abstract: This deliverable describes the actual D12 deliverable, which is a complex electronic system composed of 5 parts (modem, voltage adaptor, DSP board, host PC, power supply). The system is an experimental and demonstration bench for research and verification of novel wireless communications. Particularly, the system allows experimental verification and performance measurements of the enhanced 802.11 PHY (optimized MAC) as specified in the MobileMAN project.</p>
--

	<p>Project funded by the European Community under the “Information Society Technologies” Programme (1998-2002)</p>
---	---

CONTENTS LIST

1. INTRODUCTION	4
2. HARDWARE	5
2.1. Overview	5
2.2. FPGA content.....	6
2.3. DSP board resources.....	7
3. SOFTWARE	8
3.1. PHY Firmware	8
3.1.1 MAC Firmware.....	8
3.1.2 Packet Data Structure	17
3.1.3 Host Interface Firmware	23
3.2. Host Software.....	26
3.2.1 Drivers.....	26
3.2.2 Application.....	26
3.3. Utilities and Tools.....	26

SUMMARY

This report describes the actual D12 deliverable, a system composed of 5 parts (modem, voltage adaptor, DSP board, host PC, power supply). The system is an experimental and demonstration bench for research and verification of novel wireless communications. Particularly, the system allows experimental verification and performance measurements of the enhanced 802.11 PHY (optimized MAC) as specified in the MobileMAN project.

A detailed presentation of the code we developed and the hardware schemes can be found in a technical report published on the project web site <http://cnd.iit.cnr.it/mobileMAN/pub-deliv.html>

The Wireless Network Interface systems (4 of them have been until now manufactured) are physically located at the SUPSI campus, in Manno, Switzerland. They are available for demonstrations and inspections if requested.

1. INTRODUCTION

The aim of this deliverable is to provide a technical description of the enhanced Wireless Network Interface developed by SUPSI-DTI during the MobileMAN project. The Network Interface, a picture of which in a laboratory environment is shown in Figure 1, is in fact a multi-components system (although portable and therefore adapted for mobile use, provided enough power is supplied).

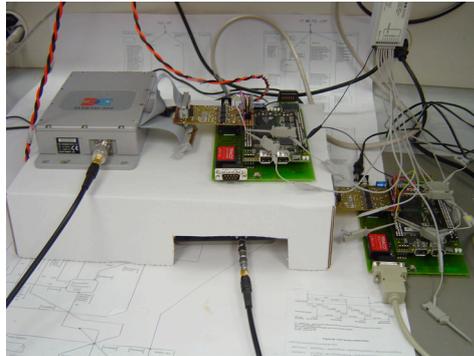


Figure 1 The enhanced 802.11 Wireless Network Interface (PHY).

The system implements a full 802.11 PHY (Radio Frequency up/down converters (RF), baseband modulator/demodulator (BB), MAC firmware on a DSP microcontroller).

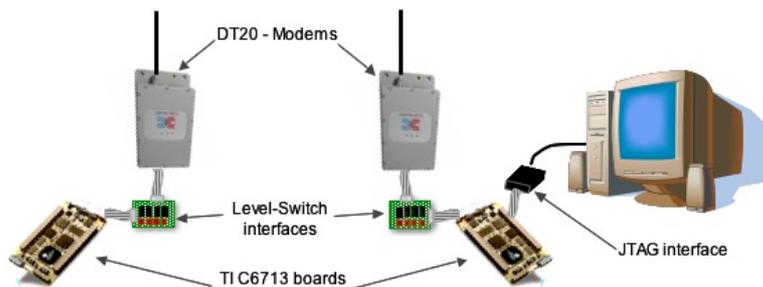


Figure 2 Typical lab setup.

The system may be used in lab environment (through development system and JTAG interface), as for instance during synthetic traffic tests. The system may also be used in more realistic environment, thanks to the high speed IEEE1394 bus which allows the full speed connection with an host PC.

RF and BB have been kept complying to the 802.11 standard; this in order to allow mixed environment experiments, where enhanced systems co-operate with standard, off-the-shelf components.

Consequently, the present report describes in details the parts proper to the MobileMAN project, namely:

- Hardware, including schematics for small adapter cards.
- Firmware running on the PHY (DSP microprocessor); this is composed of the MAC firmware, and of the host-communication firmware (allowing the host PC to transmit and receive packets to the PHY).

2. HARDWARE

2.1. Overview

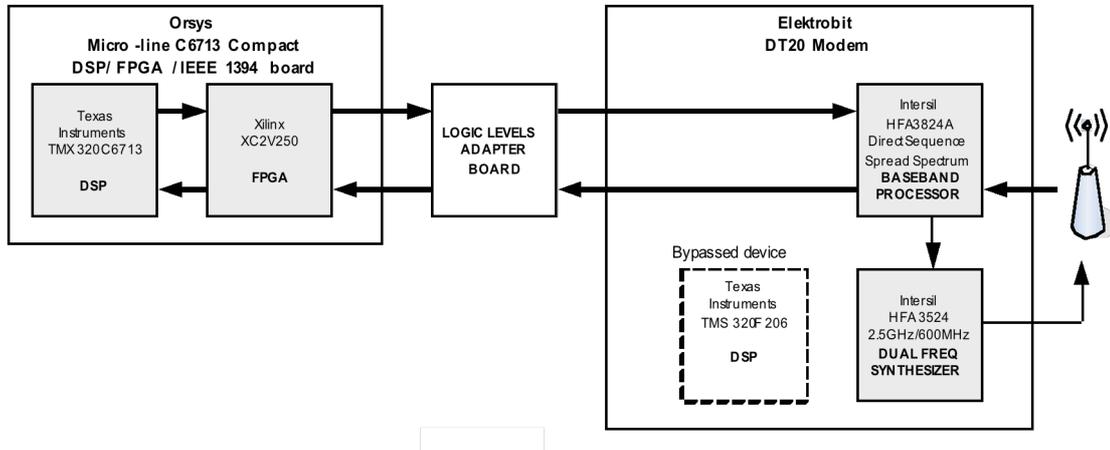


Figure 3 Overview of the enhanced 802.11 Wireless Network Interface (PHY).

The Elektrobit DT20 modem has been integrated in the system to manage the RF+BB parts of the 802.11 PHY, while an Orsys Micro-line C6713 Compact DSP board is used for the MAC implementation.

A Logic levels adapter board had to be developed to convert logic levels of the DSP board (3.3 V) to the logic levels used on of the DT20 modem (5 V).

The DT20 modem came originally with a TMS320F206 device for the HFA3824A and HFA3524 configuration and control, this is due to a previous application, which the DT20 modem was developed for. For the MobileMAN project this device is not needed, the named HFA devices are controlled by the DSP board, therefore the TMS device is bypassed by simple circuitry re-design.

The DSP board integrates an FPGA (Xilinx XC2V250) and a Texas Instruments TMX320C6713 DSP. The 802.11 MAC is implemented in standard C and specifically for the C6713 DSP, while a highly specific DT20 Modem interface had to be developed on the FPGA device, in order to establish the communication between the DSP and the modem. See Figure 4 for a more detailed overview of the interface at logic blocks level.

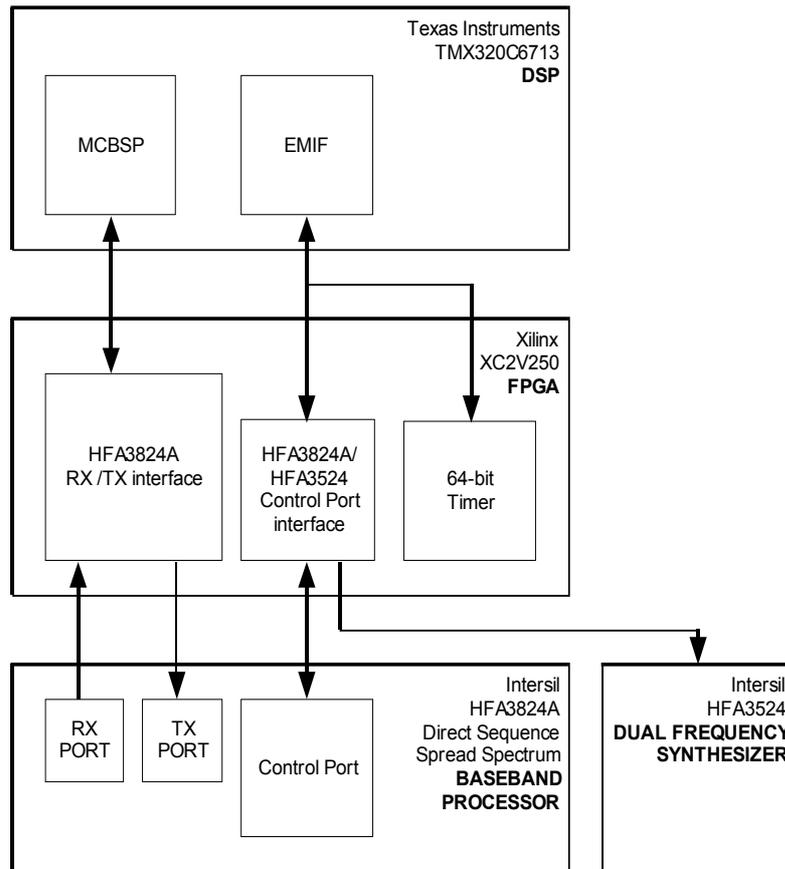


Figure 4 Logic blocks diagram of the MAC implementation. Note that only three functional blocks have been implemented in the FPGA

2.2. FPGA content

HFA3824A RX/TX Interface

This block act as glue logic between the MCBSP serial interface available on the DSP an the serial Receive & Transmit ports of the HFA3824A Baseband Processor.

HFA3824A/HFA3524 Control port interface

This block is used as interface between the DSP and the Control port of the HFA3824A device. Through this interface, the Baseband Processor and the Dual Frequency Synthesizer can be configured.

64-BIT TIMER

This is a 64-bit timer implemented for MSDU end of life management in 802.11 frame transmission and reception.

2.3. DSP board resources

The 802.11 MAC uses only a fraction of the vast amount of peripherals and resources available on the Orsys Micro-line C6713Compact DSP board. Table 1 lists the used resources and a brief explanation of the specific use for each resource is given.

Device	Resource	Qty	Use
DSP	External Interrupt Line	1	CCA assessment interrupt signal. This signal is generated by the DT20 modem (HFA3824A device) each time a low to high or high to low transition of the signal is detected.
DSP	MCBSP Serial Port	1	This high speed full duplex (feature not needed for 802.11) serial port is used to transmit and receive 802.11 frames at 2 Mbps.
DSP	32-bit Timer	2	Timer 1 : timer used for the implementation of the Backoff procedure (DIFS & slot time intervals) Timer 2 : Timeout timer for ACK and CTS frames. Note that a 64-bit timer has been implemented on the FPGA device for MSDU transmission and reception end of life time management..
DSP	Internal Interrupts	3	2 MCSBSP (receive and transmit) interrupts 1 Backoff timer interrupt.
DSP	Internal RAM	125K	125 Kbytes of DSP internal RAM is occupied by data and firmware code.
DSP	External RAM	1.2M	1.2 Mbytes of DSP external RAM is occupied by the heap (1 Mbyte, used for 802.11 received and to be transmitted data) and the FPGA code (about 200 Kbytes)
FPGA	I/O Pins used	184	92% of 200 available pins
FPGA	Gates	18203	7% of 250k available gates
Board	DT20 interface pins	20	20 pins are needed to implement the DSP-FPGA system to DT20 interface.
Board	GPIO pins	2+6	XF0, XF1 and 6 MCBSP1 pins used for debug purposes.

Table 1 DSP board used resources overview.

3. SOFTWARE

This section describes the software which is necessary in order to use the Enhanced Wireless Network Interface (PHY). Since the PHY is physically attached to the host computer through an high speed connection (IEEE1394), the 2 software packets will be described separately. We will first describe the PHY firmware (running on the on-board DSP), and then the host PC software (driver+small application).

3.1. PHY Firmware

Great effort was put in order to maintain the maximum possible abstraction at source code level for the PHY software; only few software components are specific to the C6713Compact board; among these: timing considerations, available DSP resources, configuration and control required a specific implementation (i.e. couldn't allow an high abstraction at the source code level). In other words, some software re-design must be considered in case of a change of the development platform.

The PHY firmware may be further subdivided into 2 components:

- MAC firmware. This is the hard real-time software which allow packets (fragments) to be physically transmitted and received to and from the RF interface.
- Host interface firmware. This software component is less stringent in terms of real-time requirements.
- Packet data structure. The data structure is the communication channel between MAC firmware and Host interface firmware; it is a vital part of the MobileMAN project since it allows the cross-layering functionalities between PHY/MAC and upper layers.

3.1.1 MAC Firmware

Nevertheless, the firmware comes without an operating system, which was not needed for the implementation of the standard 802.11 Frame Exchange Sequence and relative tasks (fragmentation, de-fragmentation, fragment cache control,...). This is pretty a good step in direction of a better portability of the source code.

On the actual system (C6713Compact board), the firmware occupies about 125 Kbytes and can reside completely in the DSP internal RAM, at run time.

The 802.11 MAC firmware is composed of 13 modules (source & header files). Their names and content are given in Table 2. The firmware is linked with the `c6x_boardlib`, which is a specific library for the C6713Compact board (this is a third party library implemented by Orsys).

MODULE NAME	CONTENT
block_reception (block_reception.c, block_reception.h)	<ul style="list-style-type: none"> • MCBSP receive interrupt routine • Clear Channel Assessment edge transition detection interrupt routine • Frame validation and MAC address filter • Fragment cache search/update • Defragmentation
HFA3824Adrvs (HFA3824Adrvs.c, HFA3824Adrvs.h)	<ul style="list-style-type: none"> • Primitive functions to write (configuration) & read HFA3824A control port registers. • Primitive function to configure the HFA3524 device.

Mac6713_xcv250 (Mac6713_xcv250.c, no header file)	FPGA code. The code is always downloaded on the FPGA at power-up, through the call of the FpgaLoad function (C6713Compact library) in main.c
mac_data_service (mac_data_service.c, mac_data_service.h)	Signal and preparation of an MSDU coming from the LLC
macdbg (macdbg.c, macdbg.h)	<ul style="list-style-type: none"> • RS232 command interpreter • Int to string conversion routine
macdefs (Macdefs.h, macdefs.h)	Variables and structures specific to the 802.11 MAC are defined and initialized here
main (main.h, no header file)	<ul style="list-style-type: none"> • Board, DSP peripherals, DT20 modem configuration and initialization. • Main infinite loop
mib (mib.c, mib.h)	Dot11 variables definition and initialization
mpdu_generation_STA (Mpdu_generation_STA.c, Mpdu_generation_STA.h)	<ul style="list-style-type: none"> • MSDU to MPDUs fragmentation procedure. • MPDU Queing procedures
protocol_control_STA (protocol_control_STA.c, protocol_control_STA.h)	Frame transmission state machine. (RTS-CTS-Data frame-ACK frame exchange sequence control)
timer (timer.c, timer.h)	64-bit timer, variables definition and initialization
transmission (transmission.c, transmission.h)	<ul style="list-style-type: none"> • MCBSP transmit interrupt routine • Backoff procedure
utils (utils.c, utils.h)	Utility & calculation functions

Table 2 MAC firmware components.

The 802.11 frame exchange sequence has been divided in 6 main procedures, which are explained in the following sections. Four of these procedures are interrupt routines illustrated in Table 3.

INTERRUPT SIGNAL	INTERRUPT ROUTINE	SOURCE FILE
CCA assessment (external)	PhyCcaIndication	block_reception.c
MCBSP Transmit	Data_Pump	transmission.c
MCBSP Receive	MCBSPwordReceive	block_reception.c
Backoff timer	Backoff_Procedure	transmission.c

Table 3 Interrupt routines.

3.1.1.1 Main

The *main* function (module *main*) has 2 distinct blocks of code: the *Initialization* block and the *Infinite Cycle* block. The first performs initialization tasks for DSP peripherals and the DT20 modem, the latter is simply an infinite while-loop, where the MAC waits for data to be transmitted or received.

Figure 5 illustrates all critical functions of the main module. Note the dashed blocks are debug blocks, which are not really needed at run-time, and they are used for debug and test purposes only.

A *Init MAC prms (EEPROM)* block is also present. This block can read important MAC parameters stored permanently on EEPROM, such as the station MAC address and dot11 values (like RTS threshold or fragmentation threshold). These parameters can be changed and permanently written to an EEPROM thanks to a simple utility application available at PC level.

More about that in the section dedicated to the utilities used for MAC testing and debugging.

The transmission of an MSDU is performed in the infinite cycle block, by calling fragmentation and MSDU transmission procedures explained in the next sections.

The reception of 802.11 frames is performed completely in other interrupt routines, which are also explained in the next sections.

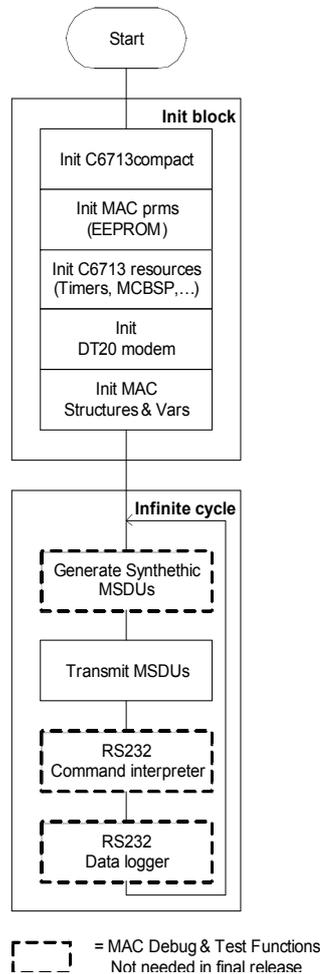


Figure 5 Flowgraph of the main module.

3.1.1.2 Transmission

Tx_Coordination function, implemented in *Protocol_control_STA* module.

The flow diagram in Figure 6 gives an overview of the 802.11 frame transmission procedure. *Tx_Coordination* is called (by main) whenever 1 or more MSDUs have to be transmitted. There are many situations where the backoff procedure is called: start of transmission, CTS timeout, ACK timeout, end of transmission. On the flow diagram, retry limits are not shown for diagram simplicity. In case these limits are reached, *TX_Coordination* exits.

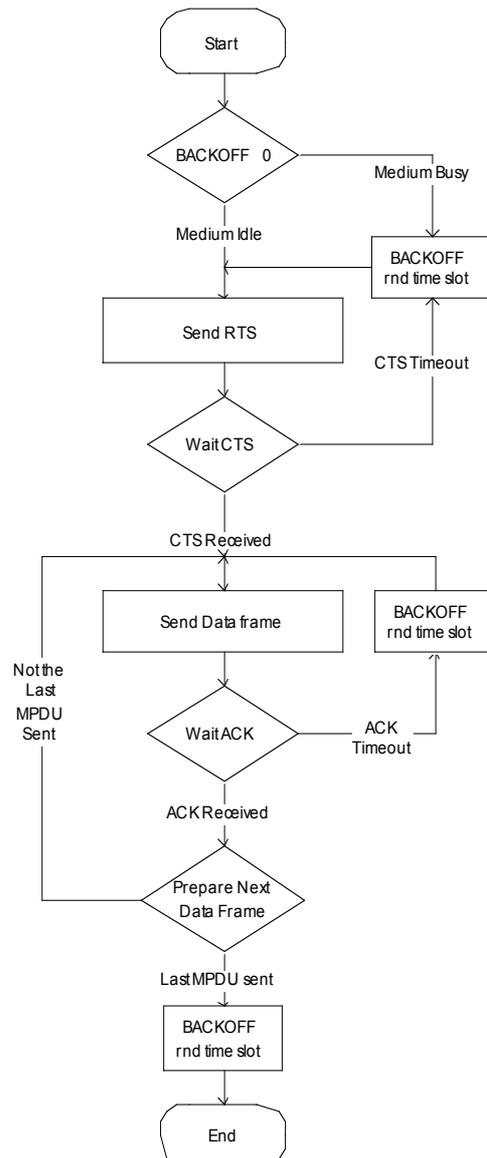


Figure 6 Flowgraph of the transmission procedure.

3.1.1.3 MCBSP Transmission

Data_Pump interrupt routine, implemented in *transmission* module.

The *Data_Pump* function is an MCBSP transmit interrupt routine, called each time a byte of a MAC frame has to be sent (the MCBSP transmit register is empty and ready to be written by the firmware). It also performs the FCS field calculation.

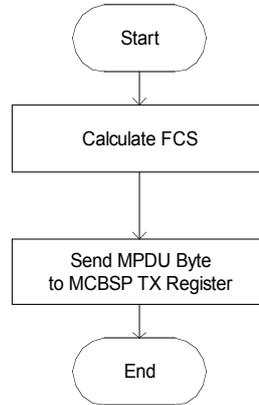


Figure 7 Flowgraph of the MCBSP transmission.

3.1.1.4 MCBSP Reception

MCBSPwordReceive interrupt routine in *block_reception* module.

The *MCBSPwordReceive* function is an MCBSP receive interrupt routine, called each time a byte of a MAC frame is received (a data byte is present in the MCBSP receive register and is ready to be read by the firmware). It performs the FCS calculation and creates the MAC frame, which will be validated in the *PhyCCAindication* interrupt routine (see next section).

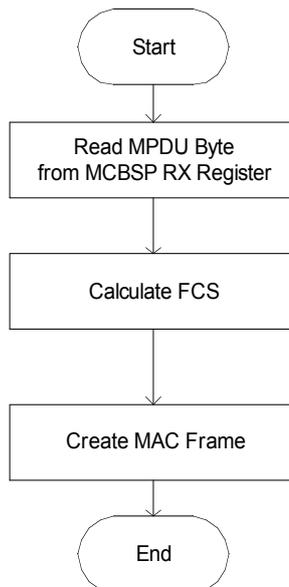


Figure 8 Flowgraph of the MCBSP reception.

3.1.1.5 CCA assessment

PhyCcaIndication interrupt routine, implemented in *block_reception* module.

This interrupt routine is called each time an edge transition of the CCA signal (generated from the HFA3824A) has been detected. The edge detection logic is implemented in the RX/TX port interface logic block on the FPGA.

The low to high transition means that energy on the transmission channel has been detected. This is not a confirmation that an 802.11 frame has been sent from a proximity station, the interrupt signal simply states that energy is on the channel, therefore reception at this point is simply initialized.

The high to low transition is detected each time energy on the channel is not present anymore, therefore an 802.11 frame has been *potentially* received from this station (many MCBSP receive interrupts are generated if the Baseband Processor has received a valid preamble/header/data frame). After this transition, the MAC starts to validate the frame (check of the FCS and protocol code) and, if frame is considered valid, it checks the STA address in order to evaluate if the received frame is for this station or not.

If these two conditions are met, the frame type must be determined and the MAC reacts to the frame type by executing different tasks. See figure 9, for more details.

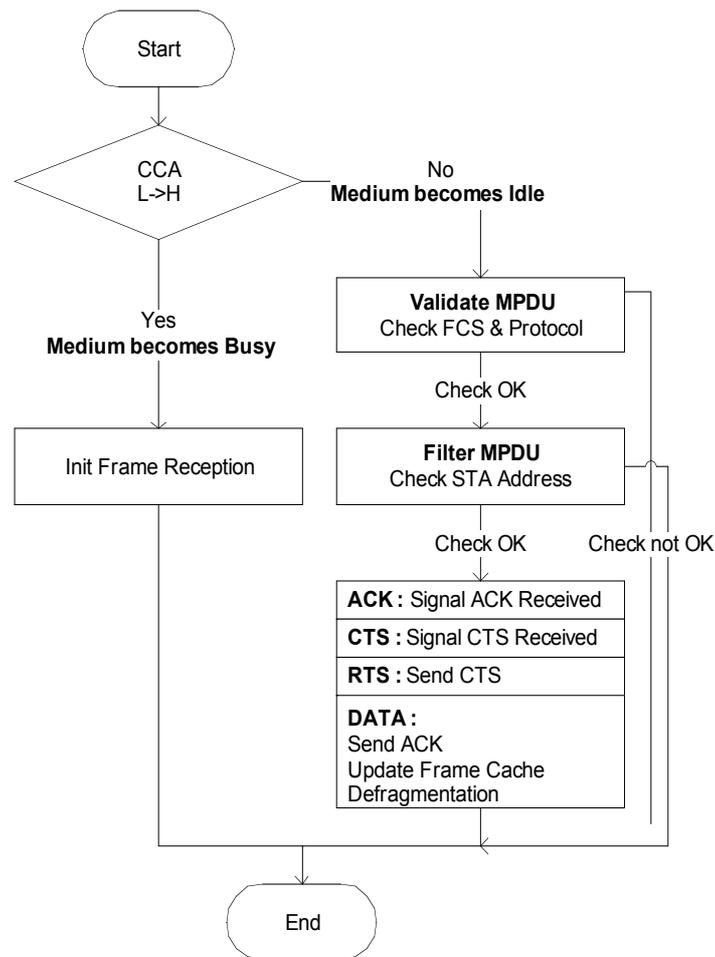


Figure 9 Flowgraph of the CCA assessment.

3.1.1.6 DT20 Modem Configuration

The most important parameters of DT20 modem operation mode are reassumed in the following table. Specific configuration for the MAC firmware is also given.

Parameter	MAC Configuration	Meaning
Communication mode	Half duplex	The modem is configured to receive-only and transmit-only operations
Transmit Preamble/Header mode	Full preamble & header Generated and processed internally	The PLCP header of a MAC frame is generated and processed by the HFA382A device (and not the DSP)
Chips per symbol	11	Number of chips per symbol used in the I and Q paths of the receiver matched filter correlators and transmit paths
Data rate & modulation (for 11 chips/bit)	2 Mbps DQPSK	Mac frames are sent and received with a 2 MBPS data rate using DQPSK modulation mode
Carrier frequency	2.447 GHz (channel 47)	Data are received and transmitted with a carrier frequency of 2.447 GHz

Table 4 DT20 modem configuration parameters.

3.1.1.7 Timing diagrams for basic frame exchange sequence

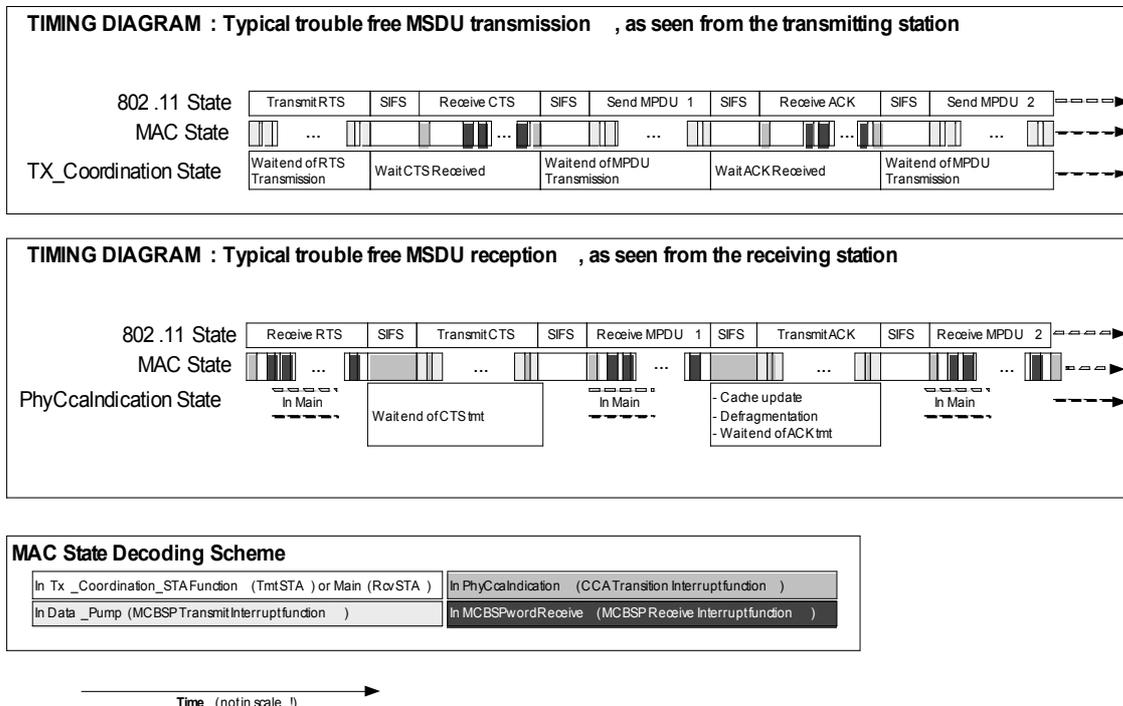


Figure 10 Timing diagrams for frame exchange sequence.

Figure 10 shows the timing diagrams (not in scale!) for a typical, trouble free (no FCS errors & receive timeouts), transmission of a fragmented MSDU (>2 MPDUs) for a transmission involving

the RTS-CTS-Data-Ack frame exchange sequence.

Two timing diagrams are given, one for the transmitting station (Tmt STA) and the other for the receiving station (Rcv STA).

For both diagrams, the 802.11 State represents what is happening at the 802.11 protocol level.

The MAC State shows in which function of the MAC firmware the DSP is running.

The *TX_Coordination & PhyCcaIndication* (both are implemented functions) states, indicate in which state of the function the DSP firmware is actually running.

In an idle state, where the Tmt & Rcv STAs are not transmitting or receiving any data, the firmware is simply running in an infinite while loop (in function main), waiting for data to be transmitted or energy on the transmission channel, indicating that some data have been potentially sent by other stations in proximity.

Tmt STA timing diagram

After preparation of the MPDUs from the fragmented MSDU, the Tmt STA exits the infinite while loop in the main function and enters the *TX_Coordination* function. It transmits an RTS frame to the Rcv STA, by sending RTS-frame bytes to the MCBSP TX Register and generating a MCBSP Transmit Interrupt (*Data_Pump* Interrupt function, coded with light grey in the timing diagram) each time a byte has been effectively sent and the TX register is ready to receive other data. The DSP remains in the Wait end of RTS transmission state until all bytes of the RTS-frame have been sent. After complete RTS transmission, the Tmt STA enters the Wait CTS Received state and it goes out from this state only after a complete reception of the named frame. Tmt STA starts to receive the CTS frame when energy is detected on the transmission channel and the CCA signal has been asserted (this generate an interrupt, which calls the *PhyCCAIndication* Interrupt function, coded with dark grey in the timing diagram). Reception of the CTS-frame ends by CCA-signal de-assertion (this also generates a call to the *PhyCCAIndication* Interrupt function). Between these 2 CCA signal transitions, the DSP enters the MCBSP receive interrupt (*MCBSPwordReceive* Interrupt function, coded with black in the timing diagram) each time a CTS-frame byte is available in the MCBSP RX Register.

Exactly the same Transmit-Wait-Receive principle is applied for the transmission of the first and subsequent MPDUs and relative reception of ACK Frames. Note that the SIFS interval is achieved by the DSP latencies involved in the firmware, no timers are used to generate this interval. These DSP latencies are guaranteed to be below the 802.11 SIFS interval specification.

Rcv STA timing diagram

The Rcv STA timing diagram follows the same principle of the Tmt STA, but with some substantial differences.

The Rcv STA is in idle state (meaning that the firmware is running in the infinite while loop implemented in main.c) until energy on the transmission channel is detected. In this example, energy is detected when the Tmt STA starts to send the RTS frame. The Rcv STA receives an interrupt request after the CCA signal generated from the DT20 modem has been asserted and enters the *PhyCCAIndication* interrupt function, thus initializing the reception of a 802.11 frame.

After reception initialization, the Rcv STA receives many subsequent MCBSP Receive interrupts (one interrupt for each byte in the RTS frame) and enters the *MCBSPWordReceive* Interrupt function as many times as the number of bytes in the RTS frame.

After complete reception of RTS and CCA signal de-assertion, the Rcv STA enters the *PhyCCAIndication* function again (this time the interrupt was generated by a high to low transition of the CCA signal) and, after SIFS, it starts to send the CTS frame. As can be seen on the timing diagram of the Rcv STA, the firmware *remains* in the *PhyCCAIndication* interrupt function, while transmitting the CTS frame (which means that MCBSP transmit interrupts are generated). Nested interrupts (the *Data_Pump* interrupt function is called each time a MCBSP transmit interrupt is generated) are therefore needed and used in this situation.

After complete transmission of the CTS frame, the Rcv STA returns in the idle state waiting for data frames.

The reception of an MPDU follows the same rules of the reception of the CTS frame, but an ACK frame must be sent instead of a CTS frame and fragment cache update and de-fragmentation tasks must be performed. Here is the most complex part of the code.

In order to avoid task and task management routines (requiring therefore an operating system) and to respect the 802.11 timing requirements (SIFS interval), the named tasks are executed *while* the Rcv STA is sending the ACK frame for a specific MPDU. Thanks to high DSP execution speed and slow 802.11b data transmission speed (2 Mbps), these 2 tasks can be executed *before* the complete ACK frame is sent and a new MPDU is transmitted from the Tmt STA. Note that, again, nested interrupts come into play.

Subsequent MPDUs sent by the Tmt STA are received following the same principle explained above for the first MPDU.

Note that in this case also, the SIFS intervals are respected just by taking advantage of DSP execution latencies.

3.1.2 Packet Data Structure

The communication between the host and the MAC board is made by the IEEE1394 interface. The connection scheme is as follow:

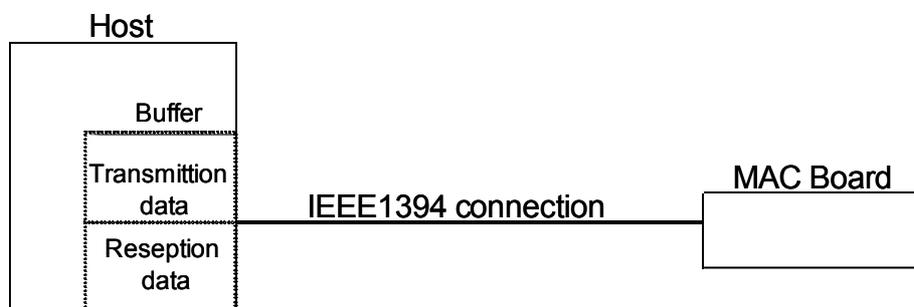


Figure 11 Connection scheme.

The communication mechanism is like Master and Slave, where the master is the MAC board. At the host layer there is a buffer memory to manage the data exchange with the MAC board that is split in to two blocks: one for the transmission data and the other for the reception data.

The data management, at the MAC board side, is achieved by a data structure that implements all the functionalities needed for the data flow management: from the host to the channel and from the channel to the host.

3.1.2.1 Packet Data Structure components

The processor unit (PU), on the MAC board, has to manage the data flow between the communication channel and the host (logical layers) in both directions and channel-to-channel in case of atomic operation (e.g. RTS/CTS). This means that the PU needs a specialized data structure. Since the communication must also satisfy the stringent time constraints and frames hierarchy imposed by the IEEE 802.11 standard, the data structure must be optimized in terms of numbers of memory access, numbers of data swap and storage mechanism.

It has been chosen to implement a buffer management scheme with a descriptor mechanism; this allows to efficiently process receive- and transmit data packets in place, and to eliminate packet copy or swapping. The buffer memory is configured in two different areas:

- data area (DA) for storing data from the logical layers to the PHY, frames ready to be transmitted, frames received from the channel and data from the PHY to the logical layers;
- transmission reception descriptor area (TRDA) for storing descriptors pointing to the data packets ready for transmission or packet received.

Thanks to the descriptor mechanism, the PU doesn't need to move, swap, copy or erase data in memory, but it has only to change some flags in the control register or to change the value of the address register that are stored in each descriptor. This data structure is easily improvable for new MAC features.

Data Area (DA)

In the Data Area are stored all the packets/frames that are flowing in all direction: from the host to the channel, from the channel to the host and from the channel to the channel. The latest case concern the atomic operation like RTS/CTS and AKN and there must to be managed immediately. For this reason there is a small amount of memory at the end of the DA, Atomic Data Area (ADA), where these frames are temporarily stored.

All the other packets/frames are stored in the Normal Data Area, (NDA), in the same order as they arrive. The storing mechanism followed by the PU is explained below:

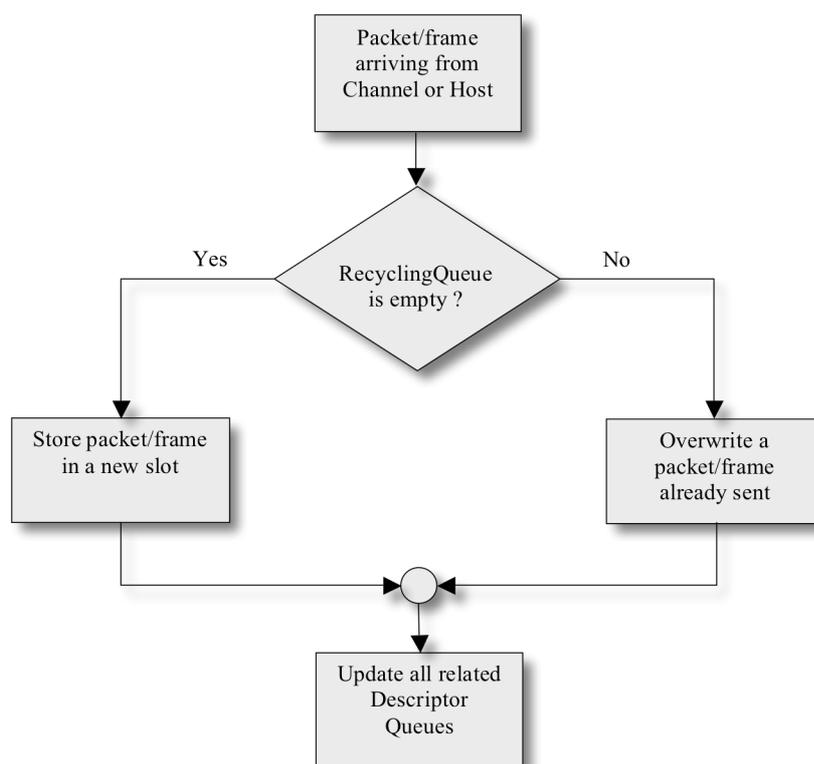


Figure 12 Storing mechanism flowchart

The length of DA is parameterized with respect to the user needs and the total memory size, and its value is stored in *MaxDALength* variable. The NDA is slotted and the length of each slot is defined by the *MaxFrameLength* variable that is defined by the user. In this way the same structure could be reused for different hardware implementations and different standards.

The NDA is also split up in two areas with different length: the bigger is the first 80% of the total NDA length and the smaller is the last 20%. The smaller area is called Guard Normal Data Area, (GNDA), and is used when the first 80% of the NDA is full. When that happens the PU knows that the memory is closed to the saturation, so it could stop the incoming data from the host until all the data prepared for the transmission on the channel has been sent.

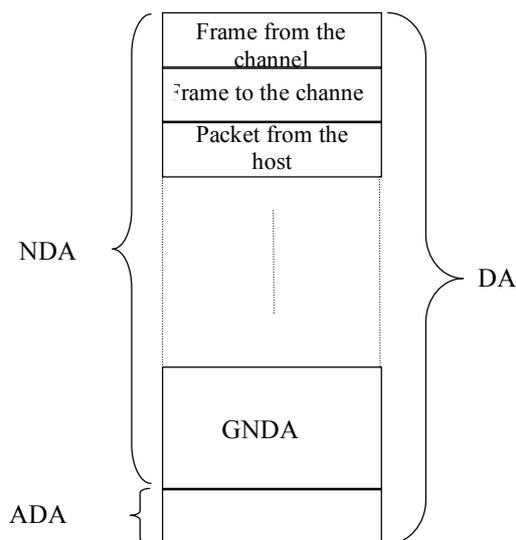


Figure 13 Data Area structure

Data storing

The NDA slots have all fixed length, which is the maximum frame length, but the data coming in the memory have variable length: variable MAC header and variable payload. Moreover, only the payload part of the packets comes from the host, since MAC header and the FCS are inserted later by the PU, above and below the payload respectively. Since the FCS is the only part that has always the same length, four bytes, the addresses for the first byte of the payload and the first byte of the MAC header are determined as follow:

$$A_{\text{payload}} = A_{\text{start_slot}} - (A_{\text{end_slot}} - (4 + N_{\text{byte_payload}})),$$

$$A_{\text{MAC_h}} = A_{\text{payload}} - N_{\text{byte_MAC_h}}$$

where,

- A_{payload} address of the first byte of the payload,
- $A_{\text{start_slot}}$ address of the beginning of the memory slot, (stored in the descriptor),
- $A_{\text{end_slot}}$ address of the end of the memory slot, (stored in the descriptor),
- $N_{\text{byte_payload}}$ number of bytes of the payload, (stored in the descriptor),
- $A_{\text{MAC_h}}$ address of the first byte of the MAC header,
- $N_{\text{byte_MAC_h}}$ number of bytes of the MAC header.

Transmission and Reception Descriptor Area (TRDA)

The transmission and reception descriptor areas are differentiated from a logical point of view, but the descriptors are stored in the same room. All the descriptors are arranged in different queues:

- host reception descriptor queue (HRDQ), from the host to the PU,
- host transmission descriptor queue (HTDQ), from the PU to the host,
- channel reception descriptor queue (CRDQ), from the channel to the PU,
- channel transmission descriptor queue (CTDQ), from the PU to the channel,
- empty descriptor queue (EDQ).

Furthermore, all the queues, except for EDQ, are organized as queue of queues where each queue is related to a certain level of priority that is previously assigned to the packet/frame, (see Figure 14). At the moment there are 7 levels of priority available, in which 0 is the higher level.

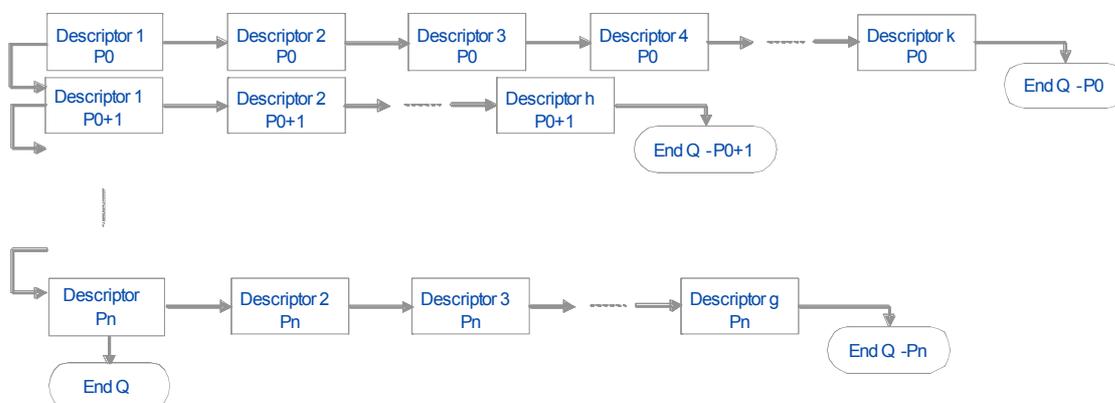


Figure 14 TDQ and RDQ structure; P is the priority level.

At the beginning of the TRDA there are 116 bytes assigned to 29 pointers, 4 bytes each one, those have the following meaning:

- 1 to 7: pointers to the beginning of the HTDQs depending on their priority level,
- 8 to 14: pointers to the beginning of the HRDQs depending on their priority level,
- 15 to 21: pointers to the beginning of the CTDQs depending on their priority level,
- 22 to 28: pointers to the beginning of the CRDQs depending on their priority level,
- 29: pointer to the beginning of the EDQ.

3.1.2.2 Data management

All the frames or data that are coming in are stored in the memory Data Area, (DA), and are consequently modified by the PU in relation of their destination.

Three main fields compose a frame: the payload, the MAC header and eventually the frame check sequence, (FCS). When the data are coming from the host they are addressed to another station somewhere in the network. In that case, the PU has to prepare the frame linking together the data, the MAC header and the result of the computation of the FCS. Vice versa, when the frame is coming from the channel, e.g. from another station, the PU has to: control the frame integrity, by checking the FCS, extract the data from the frame and eventually send it to the higher layers.

Data packet from the host to the channel

The packet coming from the host is the payload, also called frame body, and it is addressed to another station living in the network. When the packet comes it is stored in the memory Data Area, (DA). The initial address of the frame body and other information about it are stored in a descriptor in the host reception descriptor queue, (HRDQ). Before the preparation of the frame, the PU has to know the address of the target station and, if necessary, the address of a station that could forward the frame to the destination. There are two possibilities for the PU to get that information:

1. the routing information is performed by the network layer and transferred to the PU from the data link services;
2. the routing table resides at the MAC layer; in this case the PU could get the addresses directly from the routing table allocated in a special portion of its memory.

When the PU receives the routing path, it creates the MAC header and stores it immediately above the frame body. Then computes the FCS and puts the result immediately below the frame body still in the DA.

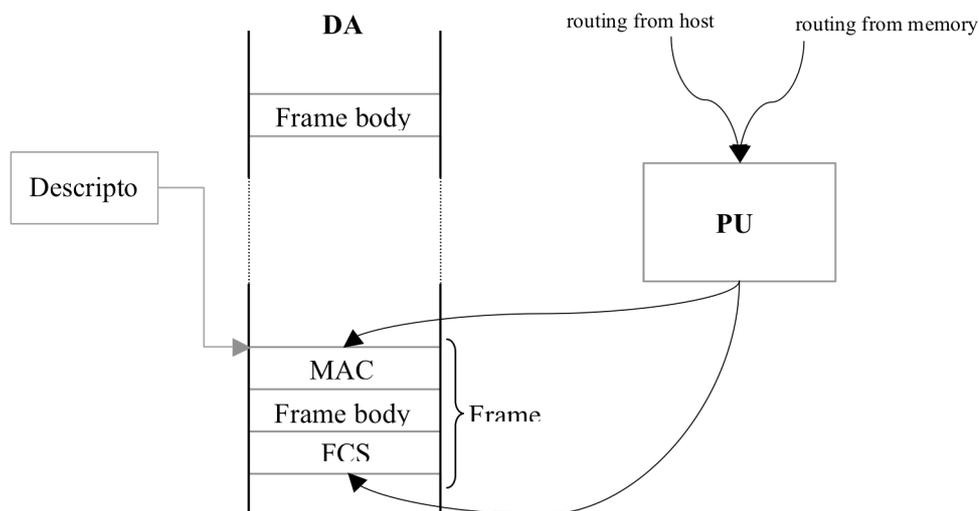


Figure 15 Data management scheme.

When the frame is complete, the PU executes the following tasks:

- put the new start address and other information about the frame in the descriptor,
- moves the previous descriptor from the HRDQ to the end of the CTDQ related to right priority,
- update all the descriptor queues registers.

At this moment the frame is ready to be sent. When the frame has been sent the PU moves the related descriptor from the CTDQ to the EDQ and updates all the descriptor queues registers.

Data packet from the PU to the host

In that case the PU acts in a reverse way respect to the previous situation:

- put the start address of the payload and other information about the frame in the descriptor,
- moves the previous descriptor from the CRDQ to the end of the HTDQ related to right priority,
- update all the descriptor queues registers.

At this moment the frame is ready to be sent. When the frame has been sent the PU moves the related descriptor from the HTDQ to the EDQ and updates all the descriptor queues registers.

3.1.2.3 Descriptors

The descriptors have all the same dimension, which is 24 bytes, and they are organized as follow:

Register name	N° Bytes	Description
Status	2	Status registers
Address_start_slot_H	2	High part address of the beginning of the memory slot pointed by the descriptor
Address_start_slot_L	2	Low part address of the beginning of the memory slot pointed by the descriptor
N_byte_payload	2	Number of bytes of the payload
Address_payload	2	Address of the payload in the slot
N_byte_MAC_heder	1	Number of bytes of the MAC header
Address_MAC_header	2	Address of the MAC header in the slot
Reserved	7	Reserved for future implementation
Address_next_descriptor_H	2	High part of the address of the new descriptor in the queue
Address_next_descriptor_L	2	Low part of the address of the new descriptor in the queue

Table 5 Descriptors content.

3.1.2.4 Global data, variables and registers

All the global data, variables and registers are defined in the `data_structure.h` file.

3.1.3 Host Interface Firmware

The host-phy interface software could be thought in three different parts: the communication driver at the PHY side (network card), the communication driver at the host side and the application still at the host side.

The drivers, at both sides, manage the connection through the IEEE1394 interface between the host and the board, so they have to implement the IEEE1394 standard communication protocol from the physical to the datalink layer.

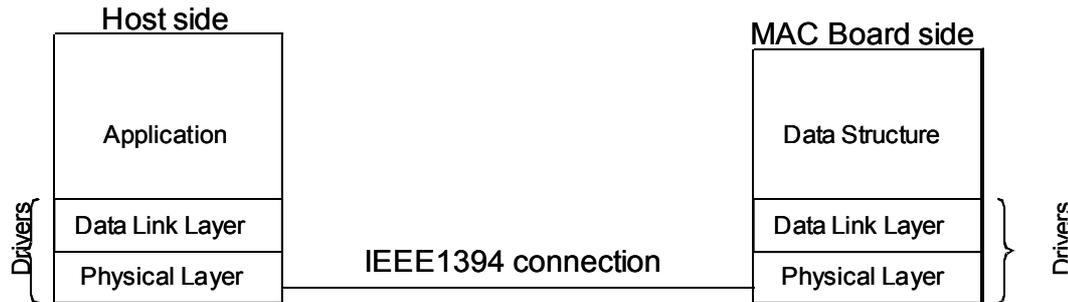


Figure 16 Software structure scheme.

Finally, the application has to handle the communication like a “higher layer”, (defined as transaction layer in the standard), over the IEEE1394 connection between the host and the board.

3.1.3.1 IEEE1394 Serial Bus Protocol Architecture

The serial bus protocols are described as a set of three stacked layers, as shown in Figure 17:

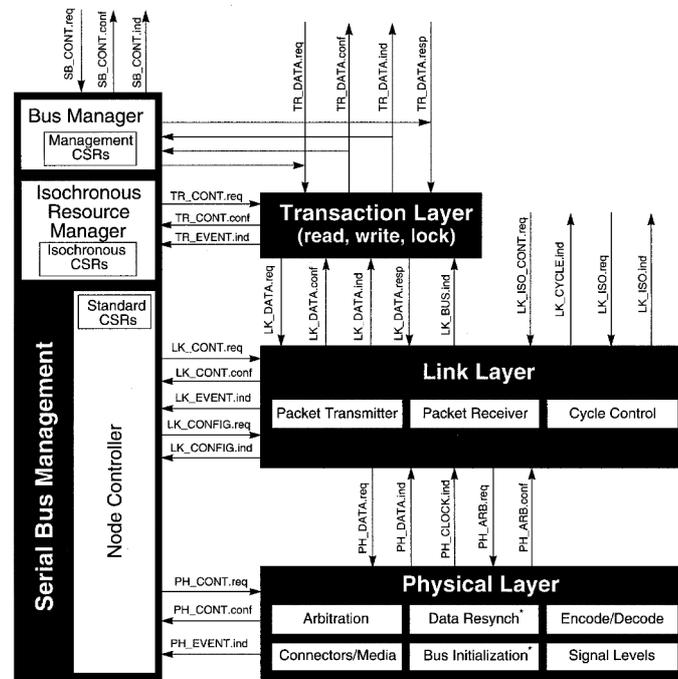


Figure 17 IEEE1394 Serial Bus Protocol Stack.

1. The transaction layer defines a complete request-response protocol to perform the bus transactions required to support the CSR Architecture (the operations of **read**, **write**, and **lock**). Note that the transaction layer does not add any services for isochronous data, although it does provide a path for isochronous management data to get to the Serial Bus management via reads from and compare-swaps with the isochronous control CSRs.
2. The link layer provides an acknowledged datagram (a one-way data transfer with confirmation of request) service to the transaction layer. It provides addressing, data checking, and data framing for packet transmission and reception. The link layer also provides an isochronous data transfer service directly to the application, including the generation of a “cycle” signal used for timing and synchronization. One link layer transfer is called a “subaction.”
3. The physical layer has three major functions:
 - 1) It translates the logical symbols used by the link layer into electrical signals on the different Serial Bus media.
 - 2) It guarantees that only one node at a time is sending data by providing an arbitration service.
 - 3) It defines the mechanical interfaces for the Serial Bus.

There is a different physical layer for each environment: cable and backplane. The cable physical layer also provides a data resynch and repeat service and automatic bus initialization.

The Serial Bus protocols also include Serial Bus management, which provides the basic control functions and standard CSRs needed to control nodes or to manage bus resources. The bus manager component is only active at a single node that exercises management responsibilities over the entire bus. At the nodes being managed (all those that are not the bus manager), the Serial Bus management consists solely of the node controller component. An additional component, the isochronous resource manager, centralizes the services needed to allocate bandwidth and other isochronous resources.

Data transfer services

This standard supports two basic data transfer services:

- a) Asynchronous data transfer
- b) Isochronous data transfer

The asynchronous data transfer service provides a packet delivery protocol for variable length packets to an explicit address and return of an acknowledgment. The isochronous data transfer service provides a broadcast packet delivery protocol for variable-length packets that are transferred at regular intervals. As shown in Figure 17, the asynchronous data transfer service uses the transaction layer, whereas isochronous data transfer service is application driven.

Transaction layer

Data is transferred between nodes on the serial bus by three different transaction types:

- a) Read—data at a particular address within a responder is transferred back to a requester.
- b) Write—data is transferred from a requester to an address within one or more responders.

- c) Lock—data is transferred from a requester to a responder, processed with data at a particular address within the responder, and then transferred back to the requester.

Transactions are multithreaded, in that a requester can start more than one transaction before the corresponding response is returned. These are called split-response transactions.

Transaction layer services

Transactions consist of four service primitives:

- a) Request—the primitive used by a requester to start the transaction.
- b) Indication—the primitive used to notify the responder of an incoming request.
- c) Response—the primitive used by the responder to return status and possibly data to the requester.
- d) Confirmation—the primitive used to notify the requestor of the arrival of the corresponding response.

These primitives and their relation to data flow is shown in Figure 18.

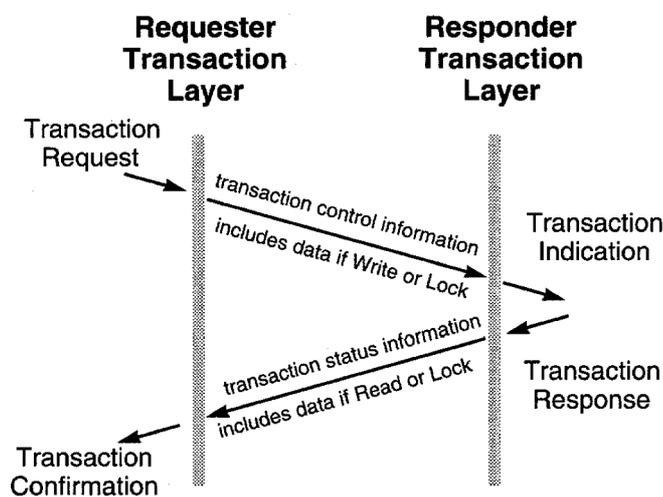


Figure 18 Transaction Services,

For more details on the IEEE1394 protocol definition please refer to the standard.

3.1.3.2 Firmware

The IEEE1394 chipset of the board, (Orsys C6713Compact), provides a direct access to IEEE1394 data through the data mover port of the link layer controller (LLC). The data mover port can operate in parallel with software accesses to LLC registers. The data are buffered by an 8Kbyte FIFO and can be routed to:

- a) the DSP via a register interface (software streaming), or
- b) the streaming port on the micro-line® connector.

Orsys already provide some simple library for the communication with the physical layer and the link layer controller.

The test program that has been used for the isochronous communication, between the board and the host, has been implemented modifying a simple example provided by Orsys as well.

3.2. Host Software

The host (PC) software has been developed under a Linux platform with a 2.4.27 kernel version. As mentioned before there are two parts on the host side: the driver and the application.

3.2.1 Drivers

The 2.4.X Linux kernel already provide some drivers for mange devices that communicate via the IEEE1394 interface. The drivers that we have used for our application are:

- a) *ieee1394*: this is the module that provides the physical layer implementation;
- b) *ohci1394*: that means Open Host Controller Interface and this is the module responsible for the communication with the controller for the link layer communication;
- c) *raw1394*: this module provides some functions, both for asynchronous and isochronous communication, that are useful for the transaction layer implementation.

The kernel has been recompiled with these three modules and consequently strongly tested before any other implementation.

3.2.2 Application

Since there are two parallel implementation that are going on (one on the PHY and one on the host PC), and since they have to communicate, the probability of doing some mistakes is very high and is very difficult to debug the whole system. So the implementation must be done step by step very carefully and each step could start only when the previews one finish with a positive test stage.

At the moment only the isochronous communication has been implemented and strongly tested while the asynchronous communication is still under development. There are two programs that implement the communication tests: *iso_control_read.c* for the acquisition of data from the board and *iso_control_write.c* for transmit data to the board.

3.3. Utilities and Tools

Texas Instruments Code Composer v2. The Texas Instruments Code Composer development environment has been used to develop, debug and test that full MAC firmware.

C6713Compact utilities & libraries. To download the MAC 802.11 firmware on the EEPROM available on the C6713Compact board, the *fload* utility is used. A serial RS-232 cable is connected between a normal PC and the DSP board and *fload* is then started.

Many tasks (FPGA download, LED status, board initialization, ...) of the 802.11 MAC are performed by calling **C6713Compact library** functions. This library (*c6x_boardlib*) is a standard item, which comes with any DSP board package.

MAC commands via RS-232. Without the Code Composer debugging environment and an emulator connected to the system, there is no way to change any parameters and/or test conditions of the MAC system (DSP board + DT20 modem) at run time. Therefore, a specific PC application has been developed to control and test the MAC when it is running as a stand alone system (without an emulator connected and TI Code Composer as control environment).

With this small and simple application, MAC parameters (for example station MAC address, signal quality thresholds, synthetic packets generation control) are fully accessible and can be changed by simply connecting a PC to the system with a RS-232 cable.

Commands to the MAC system can be fully edited and sent with specific parameters (if a new command is created, it must be also implemented at MAC firmware level, naturally).

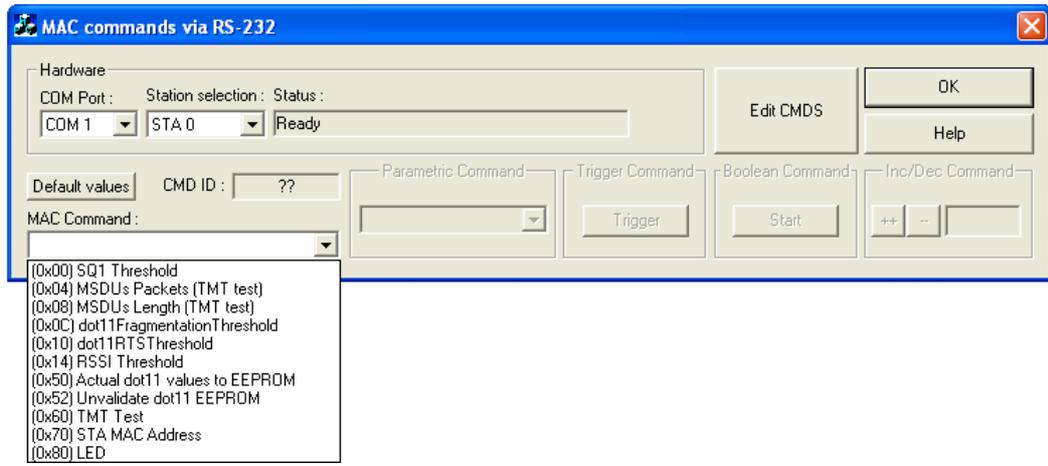


Figure 19 MAC commands via RS-232.

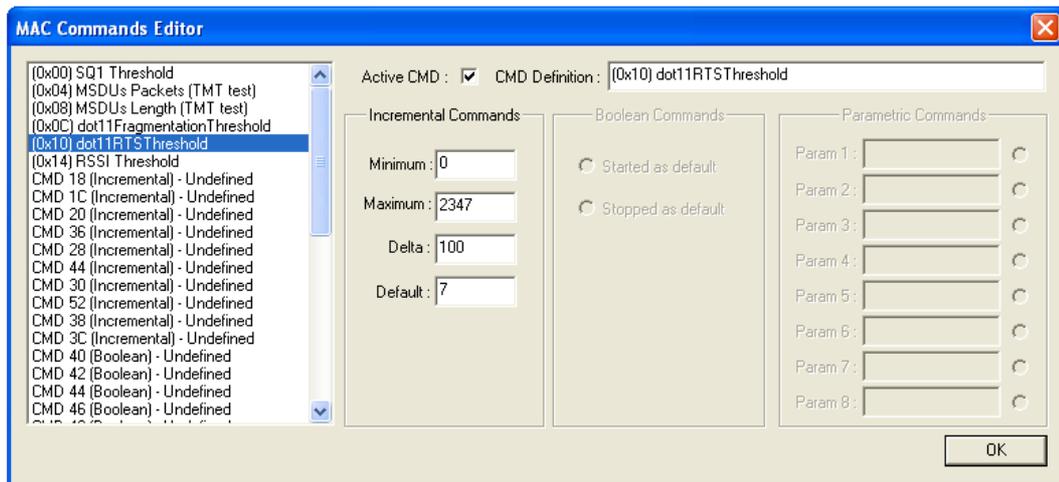


Figure 20 MAC commands editor.