A MPSoC prototyping platform for flexible radio applications

Damien Hedde
TIMA Lab
Grenoble, France
Damien.Hedde@imag.fr

Pierre-Henri Horrein
CEA-LETI
Grenoble, France
pierre-henri.horrein@cea.fr

Frédéric Pétrot
TIMA Lab
Grenoble, France
Frederic.Petrot@imag.fr

Robin Rolland
CIME-Nanotech
Grenoble, France
robin.rolland@inpg.fr

Franck Rousseau
LIG Lab
Grenoble, France
Franck.Rousseau@imag.fr

***Abstract*—Full-fledged software radio platforms are complex and expensive systems, focused on signal processing, and not very suitable for easy development and large scale experimentation. We propose a Multi-Processor System-on-Chip (MPSoC) prototyping platform targeting the support for flexible radio. This platform is fully customizable at every layer of the wireless networking stack, making it easy to prototype new protocols from the radio to the application layers. Our goal was threefold: design an efficient but cheap platform supporting flexible radio, provide support for a full system on the platform so that it can run autonomously, use "standard" components as much as possible and a modular design to ensure fast and simple development and testing to network developers. We rely on a highly modular Field-Programmable Gate Array (FPGA) based architecture. The practical results achieved so far show the effectiveness of the proposed solution in term of flexibility and cost.**

***Keywords*-Software radio, prototyping, hardware design, wireless communications, flexible radio, reconfigurable design.**

I. Introduction

With the ever-increasing demand for mobility in communication systems, wireless networking has become a major research field today. Radio systems offer mobility for various fields of communication: telephone, personal networks, business networks, long distance networks… Fields previously thought of as wired slowly become completely wireless. This leads to a wide number of standards to support all the needs and the constraints of such networks, and to an even wider number of unofficial standards, new protocols and new transmission rules. In mobile telephony, the number of standards operators are required to support is huge, with all the legacy from the early second generation and all the expected new third generation standards. Since each standard is different, sometimes even using different carrier frequency, specific stations have to be deployed and maintained, meaning very high costs and slow development. This is also true, to a lesser extent, for computer networks. Considering the pace at which new standards are being released, it quickly becomes a nightmare for anybody involved in communication systems to support them all at an acceptable cost in terms of development time and chip area. An old idea already thought of in the 70's, first developed and coined *Software Radio* by Mitola [1], was proposed to cope with such problems. With this approach, a unique device can be made compatible with a whole set of standards, for exemple ZigBee, Bluetooth, 802.11a/b/g/n, 3G, and can even be later upgraded to support new ones.

In a conventional radio device, most of the transmission chain and some parts of the Medium Access Control (MAC) are implemented in hardware, for power/performance/area/efficiency reasons, making the devices hardly evolutive. It has several consequences when working on wireless networking, either to study existing standards, or to experiment with new propositions or evolutions. Modifying the radio part is almost impossible, for example changing the TX/RX procedures, providing cutsom statistics to upper layers, or tuning the radio part from the upper layers, which is needed for cross-layer optimization [2]. The same limitations exist for the real-time part of the MAC, which is responsible for packet scheduling on the link. Many studies are then limited to the upper layers, from the application layer down to the link layer, but they can hardly touch some parts of the MAC nor the physical layer. This means that when studying wireless networking, experimentations are often limited to simulations. In a few particular cases, finding and modifying an existing device is possible, which is very time-consuming in any way, as in [3] for example. To really validate new wireless networking propositions, it is necessary to experiment with real devices in real conditions. It also usually requires a large number of devices, for example when working on sensor networks or wireless mesh networking.

A faster and easier development could be achieved by using software instead of hardware for these critical parts. When software is used to implement the whole transmission chain, the system is a Software Defined Radio (SDR) system. The main problem encountered in SDR systems is the huge processing power required. Such applications need complex computations, and in classical SDR systems, the platform used to run the software part is divided in DSPs for high-throughput processing and General Purpose Processors (GPP) for control and other computations. Such a system is hardly feasible for research purposes because of the cost and the complexity of the platform, and is thus mostly developed for industrial use only (telephony for example [4]).

In this paper, we propose a hardware/software solution

for flexible radio support that overcomes the problems explained before. With the use of a careful design, partitioned between software and programmable hardware, we provide an efficient yet fully programmable wireless device. We chose to implement the IEEE 802.11 standard for our first experimentations since it is widely available. It is also used a lot in the wireless networking community, for which easier means of experimentation are of utter interest. Its implementation is made on a low-cost platform and is completely autonomous, running it's own operating system. Thus, large scale experimentations with a large number of nodes can be envisioned at a relatively low cost. In the rest of this paper we will present the current approaches for implementing flexible radio in Section II. We then present our solution based on an FPGA and a modular MPSoC architecture in Section III. An implementation of this architecture based on a Virtex II Pro FPGA board is presented in Section IV.

II. Related work and motivations

As pure software implementations rely on ASIPs (Application Specific Instruction Processor), hybrid solutions can be found to implement flexible radio platforms, using both software and reconfigurable hardware (i.e. FPGA). Such systems are less generic than pure SDR platforms, due to the limited dynamic reconfigurability of FPGAs, yet they can be developed at lower costs, and still offer an acceptable speed of development. Dynamic reconfigurability was not uppermost in our concerns. Efficient partial reconfiguration is available in high quality (and thus expensive) FPGA only, and it was not a real issue for prototyping and experimenting new protocols. The reader interested in such platforms may read [5]. We achieve the dynamic choice of transmission by other means presented later. Many different flexible radio implementations are possible, since the limit between hardware and software can be drawn wherever we want. Some implementations, such as [6] or [7], use more hardware parts, others such as [8], are closer to pure SDR systems. The WARP platform ([7], [9], [10]) is a platform developed at Rice University, using an FPGA to implement some DSP functions, and using specific hardware. An open access repository is available, creating a growing developer community. The GnuRadio project [8] is an open-source project aiming at developing a digital signal processing (DSP) library for standard computers. It is based on a Static Data Flow representation, using elementary processing block (FFT, filters, ...) exchanging data through lossless FIFO to form higher level blocks. The project is closely associated with Ettus Research which manufactures RF Front-End boards, thus creating a full flexible radio platform. The problem with this solution lies in the need for an external computer, when a standalone board would be preferable. These solutions mostly focus their interest on one part of the whole networking stack. In contrast to these works, our goal was to develop a flexible radio device with good characteristics, being (i) *generic* and *modular*, as we want to avoid redesign and redevelopment whenever a change is made; (ii) *accessible*, to offer simplicity of development and being usable by legacy networking drivers, to allow fast and easy prototyping of new transmission schemes and protocols;(iii) *standalone* and (iv) *cheap*, to be usable with a large number of nodes and at a reasonable cost.

III. Proposed architecture

A. Networking constraints

One of our main motivations is to propose an architecture which allows prototyping of network protocols from upper to lower levels in the network stack, and experimentation from a small to a large scale (tens of nodes at least). To fulfill these constraints, it must be possible to use the proposed architecture in a complete and autonomous network structure. As a consequence, the platform must be completely autonomous, running its own operating system, driving radio interfaces for wireless communications. From our point of view, the best approach to start with is to be compatible with existing networking technologies.

These reasons led us to focus on IEEE 802.11 and to choose the MadWiFi Linux wireless device driver. MadWiFi [11] is open source project and was originally developed for Atheros chipsets. This driver is currently widely used in the networking community. The MadWiFi driver covers the network stack down to a part of the Medium Access Control layer. The part of the MAC layer that is implemented in the driver handles the management functions (like association to an access point) but not the real-time operations. The medium access state machine is de-located into the wireless device, usually a firmware programmed device whose specifications are not public.

We chose to embed a Linux based operating system into the platform in order to support the driver. As Linux is well known by most system and network developers, it provides an easy to use environment when prototyping protocols. Moreover, the large number of networking utilities provided along with Linux will allow easier debugging, and provide an easy way to collect information about the state of the wireless device.

B. High-level overview

The architecture for flexible radio we propose has several characteristics to fulfill. The most important by far is modularity and genericity, required to achieve fast development and validation of new protocols with an integration effort as low as possible. The architecture of the System on Chip (SoC) proposed for this purpose needs to take this requirement into account. To achieve modularity in the framework, we divided the architecture in independent blocks at several levels.

The highest level blocks, called macro-blocks, correspond to specified part of the network stack. This division in

functional blocks is due to the study of several wireless standards, particularly the IEEE 802.11 [12] and 802.15.4 [13] standards. Two elements of this block-based architecture are equally important: the blocks themselves, and the interfaces between the blocks. The interfaces need to be precisely defined since they provide the means to achieve modularity. As shown in Figure 1, the architecture is divided into five macro-blocks:

- Operating System (OS)
- Real-time Medium Access Control (MAC)
- Physical Layer Convergence Protocol (PLCP)
- Physical digital signal processing (PHY)
- Radio Frequency signal processing (RF)

Figure 1 shows the different macro-blocks, it also indicates the data and control interfaces between the macro-blocks.

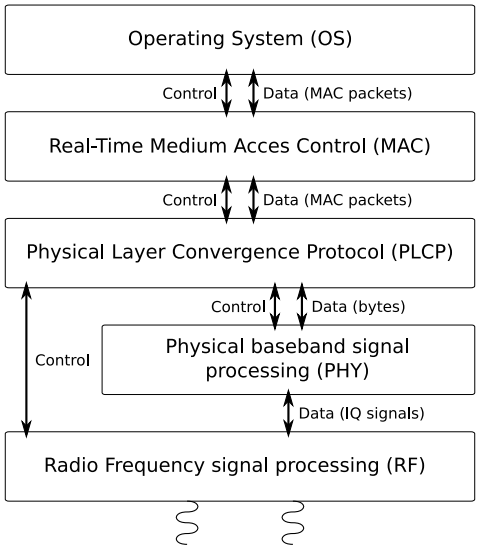

Figure 1. Global overview of the proposed architecture

The top one is the Operating System (OS) macro-block, it consists of a Linux based OS with the MadWiFi driver. The second macro-block is the Real-Time MAC, it handles the real time state machine of the MAC layer. This state machine decides when a packet must be sent in accordance with the MAC policy. The three last macro-blocks form the Physical Layer of the architecture. This layer is responsible for transmitting and receiving frames to and from the medium. It includes frame generation (and recovery) as well as modulation (and demodulation) of the radio baseband signal. Finally this layer handles the modulation (and demodulation) of the baseband signal on a carrier frequency. The Physical Layer Convergence Protocol (PLCP) macro-block of the architecture is in charge of the frame generation (and recovery). This macro-block consists in a finite state machine which controls the two remaining macro-blocks. The PHY signal processing macro-block handles the baseband signal generation (and recovery). The last macro-block is in charge of final modulation (and demodulation) on the carrier.

C. Hardware/Software mapping

As this architecture is designed to be implemented in an FPGA, the question of what is done in software and what is done using hardware logic is important. Both solutions have their own advantages and drawbacks. Indeed, a software implementation gives the most flexible solution, but with more performance limitations. On the contrary, a hardware solution offers better computing performance at the expense of flexibility.

The OS macro-block consists of the higher level of the network stack and corresponds to a Linux based operating system. This macro-block is done in software and must be executed on a processor. The next two macro-blocks (MAC and PLCP) correspond to finite state machines, each one being in a different level of the network stack. They are both real-time and may have to handle high data rates and constrained latencies. These macro-blocks are also very important because most of the flexibility in the network communication is achieved through them. Since they handle the MAC and PHY policies and mainly consist of finite state machines, we propose to map these macro-blocks into software in order to allow fast and easy modification. Figure 2 illustrates the software stack in the proposed architecture. The software stack in the OS macro-block, is divided into MadWiFi and application(s) parts. The only part that must be modified in this macro-block is the Hardware Abstraction Layer (HAL). This HAL must be adapted depending on the MAC macro-block implementation. The others part of MadWiFi can be reused without modification. This way, any evolution in the code of the driver is mostly transparent.

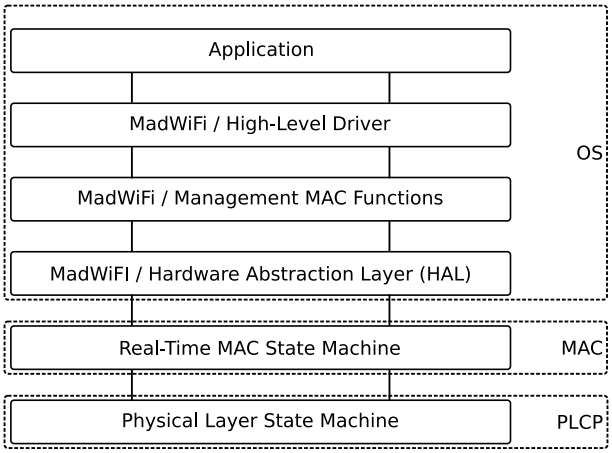

Figure 2. Software organization and mapping in the macro-block architecture

To achieve maximum flexibility, all macro-blocks should be implemented in software. However digital signal processing is very time consuming when done in software.

Since it is difficult to implement the PHY macro-block in software, we propose to design it using hardware logic. We sacrifice some flexibility for the sake of performance, but using programmable hardware keeps a good trade-off. This macro-block is further detailed in the following subsection. The last macro-block (RF) is in charge of final modulation on carrier. It cannot be done into an FPGA because of the high frequencies needed (for example, 2.4 GHz for WiFi). This block will be external to the FPGA and done using a specific chip depending on the targeted applications.

D. PHY macro-block

As said before, the PHY macro-block is in charge of the digital signal processing of the baseband signals. We propose to divide this macro-block into two signal processing chains: a transmitting (TX) chain and a receiving (RX) chain. Each is interfaced with the PLCP macro-block and RF macro-block. These chains are actually responsible for the generation (or recovery) of the baseband IQ signals from (or to) real data.

In order to design these chains, we used an approach similar to GnuRadio [8]. Yet, we decided to design these chains in hardware for performance reasons. The latency and data rate of these chain are indeed a key factor in the whole platform performance. Each chain is divided into several basic processing blocks. A processing block computes one input stream into one output stream. Examples of stream types are Byte, Bit, Phase or Complex (representing a radio signal). Special blocks are used to interface a chain with the processor and the medium front-end. The computation blocks all share the same FIFO-like interface, a block is fed by the previous block in the processing chain. Using this approach we can generate complex signal processing chains by sequencing several processing blocks. Furthermore, it provides high modularity during the design phase because processing blocks can be reused in different chains.

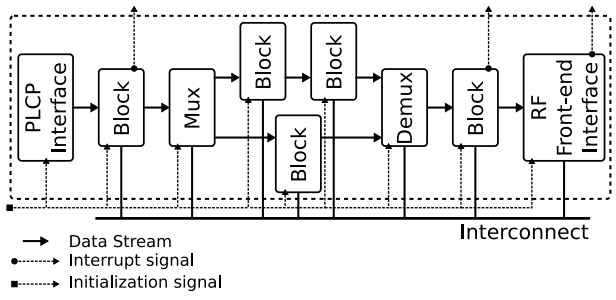

Figure 3. An example transmitting chain with two possible processing paths

Figure 3 shows the internal architecture of a processing chain. To achieve higher flexibility in processing chains, stream multiplexers and demultiplexers can be used between blocks. These components allow dynamic selection of a specific processing path in a complex chain (see Figure 3). In order to handle precisely a processing path modification, each piece of data in a processing chain is tagged with an identification label (ID). These labels are interpreted by the stream (de)multiplexer which can be configured to modify the path when a particular ID is detected. In addition to the stream interfaces, each block may have three others interfaces:

- An interrupt output signal, to signal an event to the PLCP macro-block.
- An interconnect interface, to give access to memory mapped registers.
- A compulsory initialization input signal, to trigger the initialization of the block.

The memory mapped registers are used to control (to acknowledge an interrupt for example) and configure the blocks. The configuration adds flexibility to a hardware block. The initialization signal of a block triggers a process, in which the only thing that must be preserved is the configuration of the block. In order to support a fast initialization of a whole processing chain, the initialization signals of each block of a chain are connected together and form a unique signal driven by the PLCP macro-block.

E. Interfaces specifications

Modularity is achieved through the use of independent macro-blocks in the architecture. Yet, these blocks need to be linked, and a very important part of the architecture is thus the interfaces between the different blocks. The first interface, between the MAC and OS macro-blocks is part of MadWiFi and could not be easily modified. It is not a real-time interface, but requires persistence for the data. Data may be used more than once, and since control takes place through this interface, the state of the wireless device may be shared between the two macro-block.

The next two interfaces, between MAC, PLCP and PHY macro-blocks, are critical interfaces. Since time decisions are made by the MAC macro-block, all the lower level parts of the architecture are constrained by the timing. When the decision to send data is taken, it must be done right away, not doing so would break the MAC protocol. When data are received, they must be processed immediately, to free the PHY macro-block to process the next task. Through the MAC/PLCP interface, the MAC macro-block sends and receives packets, specifies transmission configuration and gets information from the physical layer. Through the PLCP/PHY interface the PLCP sends and receives frames and controls the processing chains.

The last interface consists in the baseband and control signals needed by the RF macro-block.

IV. Implementation and results

A. The XUP Virtex-II Pro board

In order to implement the MPSoC architecture for flexible radio applications described in the previous section, we

needed to choose an FPGA board. Indeed, using an existing board has a lot of advantages. First of all, it avoids the development of a specific board, as Rice University did with WARP, which is difficult, time consuming and expensive. We decided to use the XUP (Xilinx University Program) Virtex-II Pro board. As part of the Xilinx University Program this board is very accessible for academic usage. This board is based on a Virtex-II Pro XC2VP30 FPGA, which embeds 2 PowerPC 405 core, 13000 slices, 2 Mb of Block-RAM (BRAM) memory and some hardware multipliers.

The XUP Virtex-II Pro board also provides various interface connectors. Some of those interfaces are very interesting for our platform. Using the DDR SDRAM connector we can add up to 512 MB of external memory, which is necessary in order to run a Linux Kernel. There is also a Compact Flash connector, providing an easy way to store FPGA configurations and/or file system on a flash card. The board also provides an Ethernet (10/100 Mb/s) RJ45 port as well as an RS232 serial port. It has some generic connectors too, which can be used to connect the RF front-end devices in charge of digital/analog conversions and carrier modulations in our case.

B. The 802.11 example

The 802.11 standard is the precursor of the well-known 802.11b or WiFi standard. It offers one compulsory data rate (1 Mbps), with 3 possible physical layer implementations (Infrared, Radio Frequency with Frequency Hopping, and Radio Frequency with Direct Sequence Spread Spectrum (DSSS)). Each PHY layer has an optional 2 Mbps data rate. In the following subsection, we only focus on the DSSS physical layer.

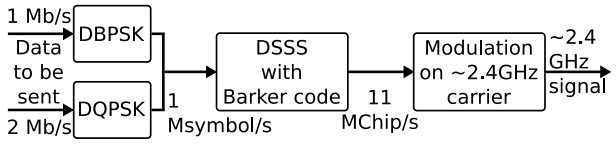

Figure 4. Simplified view of the 802.11 DSSS PHY transmitting processing

The radio frequency DSSS PHY layer operate in the 2.4GHz ISM (Industrial, Scientific and Medical) band. It uses phase modulation to represent digital data on an analog baseband (BPSK for 1 Mbps and QPSK for 2 Mbps), and uses a 11 chips word as a spread sequence. This modulation scheme is presented in Figure 4, along with required throughput at each point of the chain.

The MAC layer proposed for the 802.11 uses a well-known Carrier Sense Multiple Access / Collision Avoidance (CSMA/CA) protocol. The protocol implementation for 802.11 uses an acknowledgement (ACK) mechanism, and an optional Request To Send / Clear To Send (RTS/CTS) mechanism to avoid the hidden terminals problem. These packets (ACK, RST and CTS) are sent after a delay of duration SIFS after the packet triggering it. The collision avoidance is done using a contention window mechanism: before sending a packet, a node waits for a clear medium (duration DIFS) then waits an additional random number of clear SLOT time. Figure 5 shows the CSMA/CA mechanism without the RTS/CTS mechanism. The timings defined by the standard for the DSSS PHY layer are very restrictive:

- the SIFS time is 10 μs
- the SLOT time is 20 μs,
- the DIFS time is 50 μs, be
- the error margin for all timings is 1 μs.

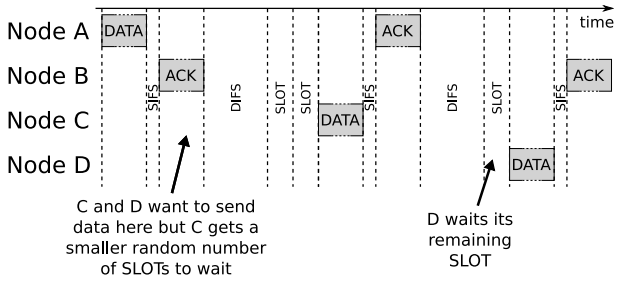

Figure 5. Illustration of the CSMA/CA mechanism

This standard, although it isn't the most complex standard to implement, is a good example of current standard as it requires small latency and multiple data rates. For these reasons, we chose it to validate the architecture concepts.

C. Implementation of the hardware/software MPSoC architecture

The Virtex-II Pro let us several choices for implementing the proposed architecture. It includes two PowerPC 405 native cores, but soft-cores can also be used the logic cells. Some MicroBlaze cores can indeed be deployed into the FPGA. The MicroBlaze is a soft-core developed by Xilinx, and is interesting because it provides several FSLs (Fast Simplex Links) interfaces, which can be connected directly to FIFOs for example. These links are available as 32-bits wide inputs and/or outputs and are accessible by using specific instructions.

Due to the reasonable processing power of the PowerPC 405 and the fact that several operating systems are available for it, we use one of the two available cores to execute the OS macro-block of the architecture. The local memory of the OS macro-block is made of 512 MB of external DDR-RAM connected to the board. Each of the remaining two software macro-blocks (MAC and PLCP) are executed on a MicroBlaze core, allowing the use of the FSLs for connecting dedicated coprocessors or interfacing with other macro-blocks. Each MicroBlaze uses its own BRAM local memory.

In our implementation, the PowerPC runs at 300 MHz and the MicroBlaze cores at 100 MHz. A good illustration

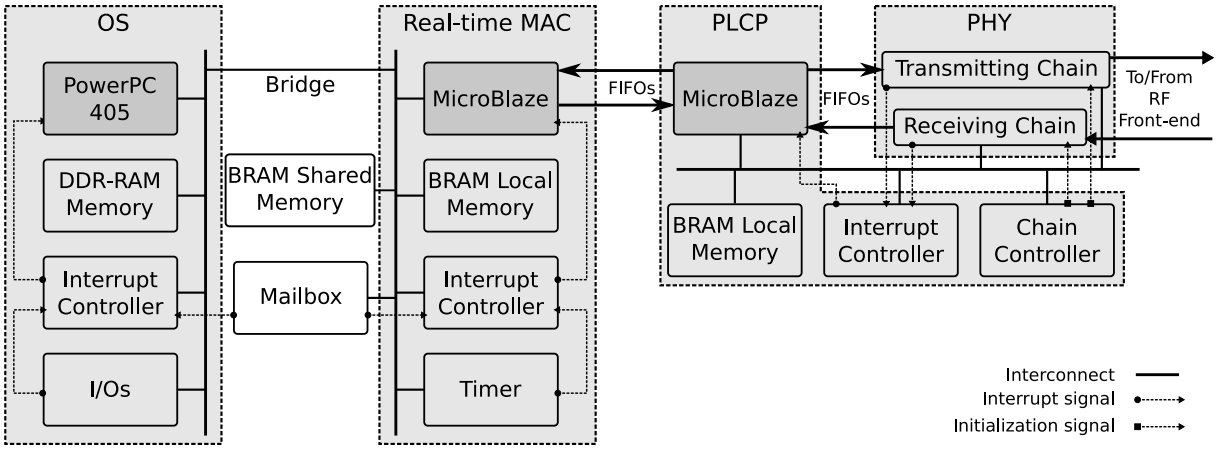

Figure 6. Implemented hardware architecture of the four upper macro-blocks

of the constraints the architecture has to cope with is the time between the reception of the last bit of a packet and the transmission of the first bit of the acknowledgment: the SIFS duration in 802.11b is 10 μs, this corresponds to 1 000 cycles when operating at 100 MHz.

Figure 6 presents the hardware implementation of the architecture, it shows only the four upper macro-blocks as the RF macro-block isn't implemented in the FPGA. The interface between OS and MAC macro-blocks uses a shared BRAM memory and a dedicated mailbox component, which uses interrupts, in order to synchronize the two macro-blocks. Packet descriptors and data reside in the shared BRAM because both macro-blocks access to them. Because these packets shall be transmitted between the MAC and PLCP under tight latency and bandwidth constraints, the MAC/PLCP interface uses FIFOs connected to the FSLs of the two MicroBlaze cores. Hence we guarantee a very small latency in the communications between the MAC and PLCP macro-blocks. We use a custom protocol between these macro-blocks, it sends predefined commands and data through the FIFO.

The interface between PLCP and PHY macro-blocks uses two different mechanisms. Each processing chain is connected to the PLCP MicroBlaze through one FIFO link, this allows the MicroBlaze to send or receive data bytes directly through these links. In a different way, the control and configuration of the chain blocks are done through memory mapped registers accessible on the local interconnect (an On-chip Peripheral Bus (OPB) in our implementation). The blocks use interrupts to signal events to the PLCP and the initialization signals of each chain are controlled through a chain controller connected to the interconnect.

The data interface between the PHY macro-block and the RF front-end is dependant on this RF front-end. The interface blocks of the processing chains must be compatible with it. Generally it consists of parallel data buses in order to send and receive data to and from analog converters. The control of the RF macro-block must be done using components connected to the local interconnect. These components depend on the RF front-end.

The Linux kernel which is executed on the PowerPC 405 is a version 2.4.26, standard networking tools are available like `ifconfig` or `iwconfig`. Because of the tight latency and bandwidth constraints, we did not put any operating system on the MicroBlaze cores and the two finite state machines are directly executed. When designing the state machines, careful management of interrupts is necessary since they can add a lot of time unpredictability. A custom HAL (see Figure 2) is used by the MadWifi driver, so that the lower level macro-blocks are seen as any MadWiFi compatible wireless peripheral.

The whole configuration of the FPGA is stored on a Compact Flash card, as well as the image of the file system which is used by the Linux kernel. Therefore the platform is standalone. This implementation also provides two external interfaces which can be used to control the platform:

- A serial interface is provided by the RS232 connector. This interface is mapped onto the main terminal of the Linux kernel, and is useful for local developments or tests of the platform.
- A network interface is provided by the Ethernet connector. This interface is also linked to the Linux kernel, and can be used to remotely control the platform as well as getting access to a remote file system.

D. Partial 802.11b physical layer implementation

In order to experiment the 802.11 DSSS physical layer (which corresponds to the lower data rates of the WiFi standard), we needed a compatible RF front-end. For this purpose we adapted the MAX19713 evaluation board to the FPGA generic connector. The MAX19713 is a Maxim analog front-end which contains several DACs (Digital to

Analog Converters) and ADCs (Analog to Digital Converters). This component has been specifically designed to handle WiFi baseband signals. The IQ baseband signals samples are transmitted through a 10 bits parallel bus at up to 45 MHz and the control of the front-end is done through a SPI (Serial Peripheral Interface) bus.

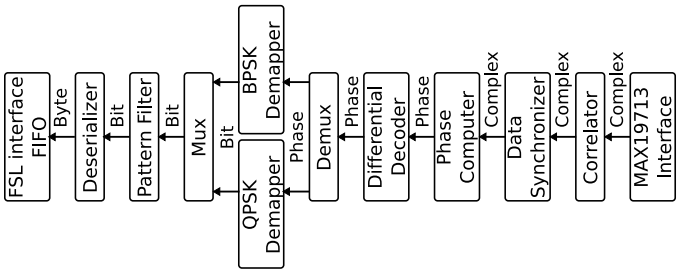

Figure 7. Receiving chain of partial 802.11b standard, the data types are described between the blocks.

We developed the PHY macro-block corresponding to the 1 Mbps and 2 Mbps data rates of the 802.11b standard. We use the MAX19713 component with a 44 MHz sampling rate, corresponding to an oversampling factor of 4. This macro-block is formed by the two TX and RX processing chains. Figure 7 presents the receiving chain. Some of the blocks are described below.

- *Correlator*: It detects the correlation of the input with the configured sequence, the output stream corresponds to the correlated results. In our case we use the Barker code of length 11: `+1+1+1-1-1-1+1-1-1+1-1`.
- *Data Synchronizer*: This block is an adapter between two clock domains, it outputs the input stream without change. It is necessary because the clock of the RF interface side of the chain is linked to the sampling rate, whereas the PLCP side of the chain is linked to the PLCP MicroBlaze clock.
- *Phase Computer*: It transforms each complex input into its corresponding phase.
- *Pattern Filter*: This block does not generate any output until a configured pattern has been detected, the input stream is then output without modification. It is used to filter the data stream until the Start of Frame Delimiter has been detected.

We didn't use any RF transceiver and connected directly the analog baseband IQ signals of two MAX19713 evaluation boards. Each MAX19713 evaluation board was connected to a XUP Virtex-II Pro board which generates its own clock (the two XUP Virtex-II boards were not synchronized with a common clock). This allowed transmission and reception of data frames at the PLCP macro-block level. Figure 8 shows an experiment using two XUP Virtex-II Pro boards and MAX19713 evaluation boards.

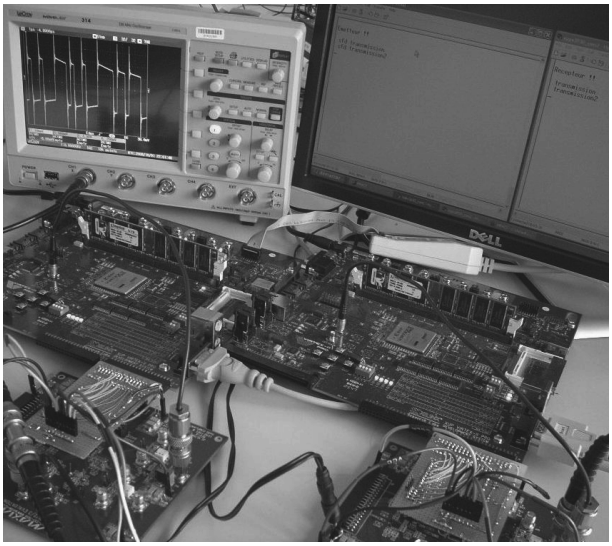

Figure 8. PHY layer implementation using the Maxim MAX19713 analog front-end

E. Validation of higher macro-blocks

Even if the platform is still under development, some results are already available. The OS block is operational, offering two running modes. A "normal processing" mode, and a development mode. The first mode optimizes the performances, as it uses a file system stored in memory, but can not be easily modified. A second mode allows easy and fast modifications of the layers, with reduced performances since it uses a remote file system. The interface between the real-time block and MadWiFi is almost completely implemented, leaving only the rates adaptation algorithms unimplemented. The real-time block implements a fully functional simplified CSMA/CA protocol, the RTS/CTS exchange is not available. Since we use the 802.11 network stack, the MAC layer handles Access Point association.

In order to validate higher macro-blocks (OS and MAC) with radio communication, we use a Cypress Wireless-USB device: the CYWUSB6935 device. This device provides an SPI interface to send and receive packet of bytes. It handles the whole physical layer of the wireless-USB, which works in the 2.4 GHz ISM band. Therefore, the PHY macro-block is inside the Cypress device. We then developed a specific PLCP state machine in order to interact with this device.

Figure 9 shows higher level macro-blocks experiments using Cypress devices. For the experiments, the device was configured with a 16 kb/s data rate. With these parameters, the real throughput for a TCP connection during a file transfer using the FTP protocol between two node using the Cypress device is 11.2 kbps. These results highlight the validity of the modular architecture as the experiment uses a 802.11 MAC layer with a Wireless-USB physical layer.

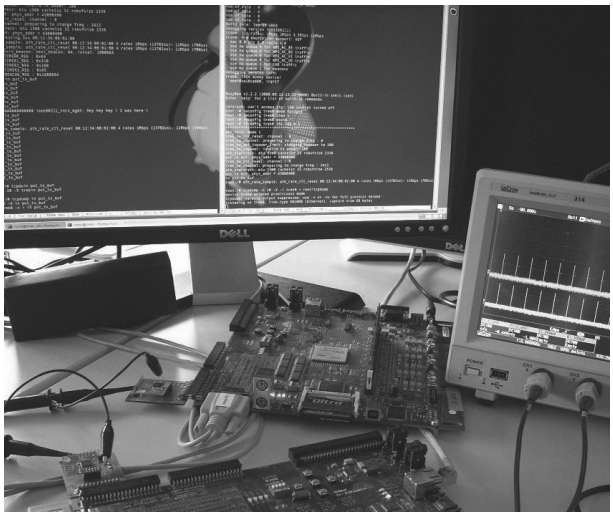

Figure 9. MAC layer implementation using the Cypress Wireless USB device

V. Conclusion

In this paper, we have presented a cheap and simple flexible radio system, based on an MPSoC architecture. The latter provides modularity and simplifies de development of time-constrained protocols. The proposed architecture offers a reduced development cycle when experimenting with network protocols. Our implementation is based on Linux, thus giving access to many standard networking tools, and on 802.11 and the MadWiFi driver, which are already widely used in the wireless networking research community. The platform can be connected to a PC during the development phase to speed-up the process, but can also be used as an autonomous wireless node: experimentations can be set up and then deployed using a large number of nodes.

So far the results are very promising, even if the developed platform is not yet fully integrated. An off-the-shelf wireless device was successfully used as a black-box PHY block interfaced under the MAC for testing purposes during its development phase. An 802.11 PHY block was developed and proved to provide the required performances. We plan to develop an 802.15.4 (ZigBee) PHY block to further validate the modular design. On the software and protocol sides, existing standard technologies were used whenever possible, drastically speeding up the coding phase for network developers.

References

[1] J. Mitola, "Software radios-survey, critical evaluation and future directions," May 1992.

[2] S. Shakkottai, T. Rappaport, and P. Karlsson, "Cross-Layer Design for Wireless Networks," *IEEE Communications Magazine*, vol. 41, no. 10, pp. 74–80, Oct. 2003.

[3] Y. Grunenberger, M. Heusse, F. Rousseau, and A. Duda, "Experience with an implementation of the Idle Sense wireless access method," in *Proceedings of ACM CoNEXT*, Oct. 2007.

[4] U. Ramacher, "Software-Defined Radio Prospects for Multistandard Mobile Phones," *Computer*, pp. 62–67, October 2007.

[5] J.-P. Delahaye, J. Palicot, C. Moy, and P. Leray, "Partial Reconfiguration of FPGAs for Dynamical Reconfiguration of a Software Radio Platform," in *Proceedings of IST Mobile and Wireless Communications Summit*, July 2007.

[6] A. Saha and A. Sinha, "Radio Processor – A New Reconfigurable Architecture for Software Defined Radio," in *Proceedings of International Conference on Computer Science and Information Technology*, 2008, pp. 709–713.

[7] P. Murphy, A. Sabharwal, and B. Aazhang, "Design of WARP: A Wireless open-Access Research Platform," in *Proceedings of European Signal Processing Conference*, September 2006.

[8] G. FSF, "GNU Radio – GNU FSF Project. http://www.gnu.org/software/gnuradio."

[9] K. Amiri, Y. Sun, P. Murphy, C. Hunter, J. R.Cavallaro, and A. Sabharwal, "WARP, a Unified Wireless Network Testbed for Education and Research," in *Proceedings of IEEE International Conference on Microelectronic Systems Education*, June 2007, pp. 53–54.

[10] C. Hunter, J. Camp, P. Murphy, A. Sabharwal, and C. Dick, "A Flexible Framework for Wireless Medium Access Protocols," in *Proceedings of Asilomar Conference on Signals, Systems and Computers*, 2006.

[11] MadWiFi, "MadWiFi Project – Trac. http://madwifi-project.org/."

[12] *IEEE Std 802.11, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, Institute of Electrical and Electronic Engineers, June 2007.

[13] *IEEE Std 802.15.4, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*, Institute of Electrical and Electronic Engineers, October 2003.

[14] T. Shono, "IEEE 802.11 Wireless LAN Implemented on Software Defined Radio With Hybrid Programmable Architecture," *IEEE Transactions on Wireless Communications*, vol. 4, pp. 2299–2308, September 2005.

[15] A. Tribble, "The software defined radio: Fact and fiction," January 2008, pp. 5–8.