Prototyping of a WLAN system using C++ based Architecture Exploration

Nikolaos S. Voros
Dept. of Communication Systems and Networks
TEI of Mesolonghi
Ethniki Odos Antiriou Nafpaktou, Varia
Nafpaktos 30300, Greece
voros@teimes.gr

Konstantinos Masselos
Dept. of Computer Science
and Technology
University of Peloponnese
Terma Karaiskaki, Tripolis, 22100, Greece
kmas@uop.gr

ABSTRACT
Wireless Local Area Networks (WLANs) are currently considered as one of the most popular application domains. In this paper the protyping of a WLAN system on a platform including microprocessors and FPGAs is described. The prototyping started from architecture exploration using a C++ library for hardware/software codesign. The developed prototype allowed evaluation of the system performance and the architecture decisions. The use of the systematic architecture exploration allowed making correct decisions early in the design cycle thus avoiding time consuming iterations from lower design stages (necessary when constraints are not met by the final implementation).

Keywords
Wireless systems, architecture exploration, hardware/software codesign.

1. INTRODUCTION
There have been several standardization efforts in wireless communications (including GPRS, EDGE, and UMTS), aimed at meeting the expected increased requirements of users and applications. In addition, and complementary to the mobile telephony data transmission standards developed in Europe, Japan and US standards for Wireless Local Area Networks (WLANs) in the 2.45 GHz and 5 GHz bands have been developed as well. In the unlicensed band of 2.45 GHz the IEEE 802.11b [1] standard has provided to the users up to 11 Mbit/s transmission rates. The IEEE 802.11a [2] and the HIPERLAN/2 [3] protocols were specified to provide data rates of up to 54 Mbit/s for short-range (up to 150 m) communications in indoor and outdoor environments.

The volumes of WLAN components (Access Points and Network Interface Cards) shipments imply implementation of WLAN systems as single integrated circuits (Systems-on-Chip) for cost reduction. This is more important for Network Interface Cards that have larger volumes and tighter size constraints. Prototyping on platforms with discrete components, such as microprocessors and FPGAs, is an important step in the development of a System-on-Chip since it allows efficient functional and performance verification of the design and validation of major architecture decisions.

Mobimedia'07, Month 8, 2007, Nafpaktos, Aitolokarnania, Greece.

In this paper the prototyping of the HIPERLAN/2 system on the ARM integrator platform [3] is described. ARM integrator includes a number of ARM processor based modules and a number of FPGA based modules organized around an AMBA bus located in the main board. The prototyping allows emulation of the finally targeted System-on-Chip implementation, offers a test-bed for the functional debugging of the developed protocol-stack, allows algorithmic performance evaluation (for the baseband part) through field measurements and validation of the architecture decisions (also in view of the final System-on-Chip implementation).

The prototyping of the HIPERLAN/2 system on the ARM integration platform is guided by systematic architecture exploration based on C++. The adoption of this systematic architecture exploration approach allows making correct architecture decisions early in the design cycle. This eliminates time consuming iterations from lower level design stages (to refine architecture in case where performance and cost constraints are not met).

The rest of the paper is organized as follows: Section 2 provides an overview of the HIPERLAN/2 system. Section 3 provides details on the functional specification of the HIPERLAN/2 system, while Section 4 details the architecture exploration phase, which has been based on OCAPI-xl environment. Section 5 presents concrete results for the final system implementation, and Section 6 concludes by providing a summary of the proposed approach.

2. OVERVIEW OF HIPERLAN/2 WLAN SYSTEM
The HIPERLAN/2 system [4, 5, 6] is composed of two types of devices: the *Mobile Terminals* (MT) and the *Access Points* (AP). Typical architectures of the Access Point and the Mobile Terminal are presented in Figure 2.

The HIPERLAN/2 basic protocol stack and its functions are shown in Figure 3. The convergence layer (CL) offers a service to the higher layers. The DLC layer consists of the Error Control function (EC), the Medium Access Control function (MAC) and the Radio Link Control function (RLC). It is divided in the data transport functions, located mainly on the right hand side (user plane), and the control functions on the left hand side (control plane).

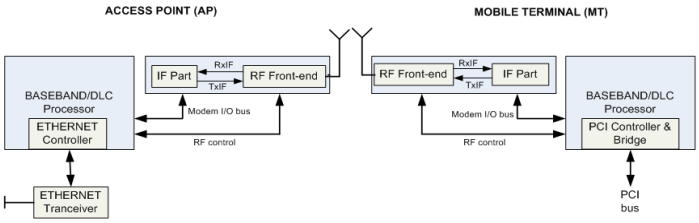

Figure 2. Architectures of AP – MT

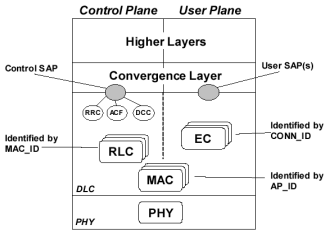

Figure 3. HIPERLAN/2 protocol stack and functions

The medium access control (MAC) is a centrally scheduled TDMA/TDD scheme. It consists of a sequence of MAC frames of equal length with 2 ms duration. Each MAC frame consists of several phases: Broadcast (BC) phase, Downlink (DL) phase, Uplink (UL) phase, Direct Link Phase (DiL), Random access phase (RA).

The MAC/DLC layer is basically structured in two levels of computing:

- A lower level dedicated to critical timing tasks: medium access, acknowledgement, timestamp, CRC, encryption, etc.
- An upper level dedicated to less critical timings tasks like fragmentation, association, authentication, device management, data transfer to/from the host, power save queuing etc.

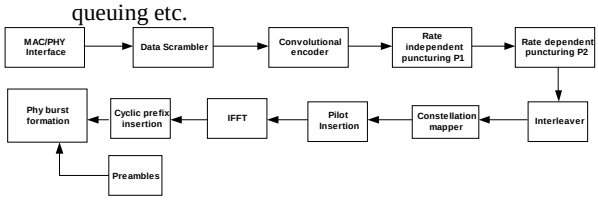

Figure 4. HIPERLAN/2 transmitter block diagram

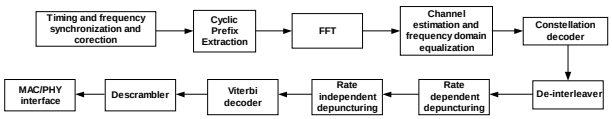

Figure 5. HIPERLAN/2 receiver block diagram

The baseband part of the physical layer of HIPERLAN/2 is based on orthogonal frequency division multiplexing (OFDM) [7]. The physical layer provides several modes with different coding and modulation schemes, which are selected by link adaptation. BPSK, QPSK and 16QAM are the supported subcarrier modulation schemes. Furthermore, 64QAM can be used in an optional mode. Reference block diagrams of HIPERLAN/2 transmitter and receiver are presented in Figure 4 and Figure 5 respectively.

Table 1. Computational complexity of transmitter tasks in different physical layer modes

		Computational complexity (MOPS) / PHY mode (Mb/s)						
Task	**Type of processing**	**6**	**9**	**12**	**18**	**27**	**36**	**54**
Cyclic prefix extraction	*Word level – memory accesses*	96	96	96	96	96	96	96
Frequency error correction	*Word level – multiplications, additions, memory accesses*	208	208	208	208	208	208	208
FFT	*Word level – multiplications, additions, memory accesses*	922	922	922	922	922	922	922
Frequency domain equalization	*Word level – multiplications, additions, memory accesses*	132	132	132	132	132	132	132
Constellation demapping	*Group of bits – LUT accesses*	48	48	240	240	288	288	336
Deinterleaving	*Group of bits – LUT accesses*	48	48	96	96	192	192	288
Depuncturing (Rate dependent)	*bit level – logic operations*	0	50	0	99	118	198	297
Depuncturing (Rate independent)	*bit level – logic operations*	0.16	0.20	0.16	0.20	0.28	0.20	0.20
Viterbi decoding	*Bit level I/O - word level additions, comparisons*	1170	1755	2340	3510	5265	7020	10530
Descrambling	*bit level - shift register, XOR*	108	162	216	324	486	648	972
Sum		2732	3421	4250	5627	7707	9704	13781

Table 2. Computational complexity of receiver tasks in different physical layer modes

		Computational complexity (MOPS) / PHY mode (Mb/s)						
Task	**Type of processing**	**6**	**9**	**12**	**18**	**27**	**36**	**54**
Scrambling	*bit level – shift register, XOR*	108	162	216	324	486	648	972
Convolutional encoding	*bit level – shift register, XOR*	174	261	348	522	783	1044	1566
Puncturing (Rate independent)	*bit level – logic operations*	0.31	0.31	0.31	0.31	0.31	0.31	0.31
Puncturing (Rate dependent)	*bit level – logic operations*	0	33	0	66	105	132	198
Interleaving	*Group of bits – LUT accesses*	48	48	96	96	192	192	288
Constellation mapping	*Group of bits – LUT accesses*	30	45	36	54	54	72	90
Pilot insertion	*Word level – memory accesses*	56	56	56	56	56	56	56
IFFT	*Word level – multiplications, additions, memory accesses*	922	922	922	922	922	922	922
Cyclic prefix insertion	*Word level – memory accesses*	72	72	72	72	72	72	72
Sum		1410	1599	1746	2112	2670	3138	4164

Both the transmitter and the receiver include tasks with different granularities with respect to the bitwidth of their operands. The tasks can be broadly classified into *fine grain* tasks that operate on groups of small number of bits and, to *coarse grain* tasks that operate on words (groups of larger number of bits). Different types of data manipulating operations are performed by the different tasks. Coarse grain tasks perform

mainly arithmetic operations (e.g. multiplications, additions) on their input data. Fine grain tasks perform logic level operations and some simple arithmetic operations. The types of operations performed by the different tasks and their computational complexities for all the different physical layer modes are presented in Table 1 and Table 2. Indicative computational complexities are given in MOPS (assuming that all operations e.g. arithmetic, logical are treated the same). However, taking into consideration the main type of processing given in second column, the real complexities become more clear. Some of the tasks have complexities that remain stable for all the physical layer modes (e.g. FFT), while some other tasks have complexities dependent on the physical layer mode (e.g. convolutional encoding, Viterbi decoding).

Part of the timing critical MAC/DLC functionality is associated with timed state machines (e.g. medium access). Some other tasks such as encryption (can be possibly included in DLC functionality) require some arithmetic processing.

3. FUNCTIONAL SPECIFICATION OF HIPERLAN/2 SYSTEM

The functionality of the targeted system has been specified through the development of an executable model in ANSI C, for the MAC sub-layer and the baseband part of physical layer. The size of the model is 20000 lines of code. The structure of the ANSI C model is shown in Figure 6.

The model of the baseband processing part of HIPERLAN/2 system is divided into two parts: *complex numbers based algorithms* (mapping, OFDM, PHY bursts) and *binary algorithms* (scrambling, FEC, interleaving).

A number of configuration parameters are supported for the physical layer modules:

- width and position of point in fixed point, numbers (separate for frequency domain, time domain, FFT calculations, FFT twiddle factors, channel correction and CFO cancellation multipliers),
- number of soft bits in Viterbi algorithm soft value representation,
- time synchronization threshold, duration and time-outs,
- the highest confidence level threshold of the de-mapper,
- sizes of internal buffers (FFT buffers, receiver command buffer, receiver data buffer).

The submodules of the physical layer are implemented as procedures, which get as standard parameters: request type, command, command parameters and data. Shared data are represented as global variables. Each submodule has a global variable, the value of which indicates the procedure where output will be directed. By default, the value of this variable corresponds to the next module in the physical layer hierarchy. Each physical layer module calls the next one, when the data portion requested by the next module interface is ready. Control information (commands) is forwarded synchronously with data, except of shared FFT modules and Viterbi algorithm internals.

Significant part of the high level design of the MAC layer is common for Access Point and Mobile Terminal devices. The high-level design of the MAC layer focuses on external interfaces of the sub-layer. In contrast to physical layer, MAC layer's module intercommunication is activated when a logically finished data structure is completely ready. Information is transferred in the form of memory pointers, or copied to some buffer.

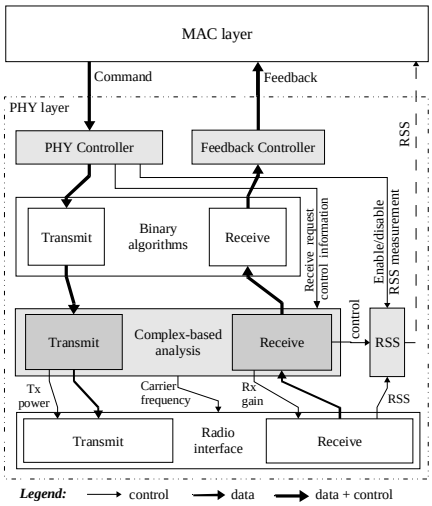

Figure 6. Structure of the ANSI C model of the targeted functionality

4. ARCHITECTURE EXPLORATION

Place Architecture exploration is related to the stages of the design flow that link high level specifications with implementation detailed design steps (HDL coding for hardware, C/C++ coding for software). The major decisions made during architecture exploration include the allocation of different types of processing resources and the assignment of the targeted functionality tasks on the allocated resources. Given the complexity of modern applications, making such decisions in a non systematic way (i.e. based on designer's experience) and with no tool support leads, in most cases, to implementations that either do not meet the system's constraints, or are cost inefficient, or both. Systematic architecture exploration methods are essential to ensure correct architecture decisions early enough in the design flow, in order to eliminate the time consuming iterations from low level design stages in the cases where performance and cost constraints are not met. There are basically two types of architecture exploration approaches: the *tool oriented* design flow and the *language oriented* design flow. Example of a tool oriented design flow is the N2C by CoWare [8]. In the design flows supported by such tools, the refinement process of a design from unified and un-timed model towards RTL is tool specific. Examples of language oriented design flows are OCAPI-xl [9] and SystemC [10]. Language based approaches are more flexible and give the designer more freedom.

4.1 The OCAPI-xl environment

The architecture exploration for the HIPERLAN/2 system has been performed using OCAPI-xl C++ library [9]. OCAPI-xl is a C++ based design environment for development of concurrent, heterogeneous HW/SW applications. The

computational model of OCAPI-xl relies on processes. OCAPI-xl supports the following types of processes:

- *High Level Software processes (procHLSW).* Typically these are the processes that constitute the initial system model in OCAPI-xl.
- *High Level Hardware processes (procHLHW).* In these processes, time progresses by fixed amounts equal to the clock cycle period.
- *Managed Software processes (procManagedSW).* This type of OCAPI-xl processes allows processes to be sequentialized. In order to create a process of type *procManagedSW*, the designer must create a scheduling object. This scheduler will perform the actual sequentialization of all the processes.

Other types of processes are also provided to allow the refinement of the model, not only from a functional point of view, but also with respect to the behavior in the time domain. More specifically, OCAPI-xl supports:

- *procANSIC* processes, which are refinements of procHLSW and procManagedSW types of processes; they allow C code generation.
- *procOCAPI1* processes, which are refinements of procHLHW processes and allow VHDL/Verilog code generation.

4.2 Architecture exploration of the HIPERLAN/2 system

Using the ANSI-C model as input, OCAPI-xl models of the HIPERLAN/2 MAC sub-layer and the baseband part of the physical layer have been developed. For the high level exploration, the high-level OCAPI-xl processes (procHLHW, procManagedSW and procHLSW) have been used to model the timing behavior of the HIPERLAN/2 tasks under different abstract implementation scenarios (on generic hardware and software processors). For the software computation models a simple processor model (resembling the targeted ARM architecture) has been developed. The system partitioning and mapping to processors approach is simple. The different tasks of the targeted system are assigned to different (hardware or software) abstract processors. The abstract implementations are evaluated using system performance estimates (in terms of execution cycles) obtained by OCAPI-xl and through area cost estimates. Using this approach, different mappings of HIPERLAN/2 tasks on hardware and software have been evaluated and the most promising solution has been identified.

The process followed for partitioning and mapping on hardware and software is detailed in the sequel through two simplified examples, based on the system models of the HIPERLAN/2 Access Point system.

Physical layer

Eight critical processes of the transmitter are considered: *mapper*, *interleaver*, *Inverse FFT*, *puncturing*, *scrambler*, *convolutional encoder*, *code terminator*, *feedback controller*. One of the operational scenarios considered during architecture exploration concerns the transmission of four PDU trains (one SCH and three LCH) from the access point to the mobile terminal. The timing constraint for this operation according to the standard is 254 μs.

At the beginning of the partitioning/mapping process all the processes are modeled as procHLSW corresponding to an implementation with eight parallel software processors (ARMs), one for each process. The simulation of the OCAPI-xl model gives an estimate of 1242881 cycles for the completion of this operation. Assuming a conservative clock frequency of 50 MHz (cycle 20 ns) for the ARM processors, we get a time estimate of 24857.62 μs which is far greater than the 254 μs constraint. It must be noted that if all processes are modeled as procManagedSW, corresponding to an implementation on a single software processor shared among the processes under the control of an operating system, the execution time estimates would be far worse. Except from the timing issue, an implementation with 8 software processors is also cost inefficient. In the next steps of the exploration the processes are moved one after the other to hardware accelerators (modeled as procHLHW) leading to execution time speed up and also cost reduction (since it is reasonable to assume that the cost of an accelerator is smaller than the cost of the generic software processor corresponding to ARM). When all processes are moved to hardware the estimated number of execution cycles is 12348 leading to an estimated execution time of 246.96 μs (assuming clock frequency of 50 MHz) which is within the timing constraint. Thus we can conclude that the given processes need to be accelerated and assigned on hardware. The detailed results of this process are presented in Table 3.

Table 3. Part of architecture exploration for the baseband processing part of HIPERLAN/2 using the OCAPI-xl approach

	mapper	Inter-leaver	IFFT	Punctu-ring	Scram-bler	Conv. encoder	Code Termi-nator	Feed-back Contro-ller	Perform-ance in cycles	HW Cost
1	HLSW	HLSW	HLSW	HLSW	HLSW	HLSW	HLSW	HLSW	1242881	8 processors
2	HLHW	HLSW	HLSW	HLSW	HLSW	HLSW	HLSW	HLSW	1242188	7 processors + 1 HW accelerator
3	HLHW	HLHW	HLSW	HLSW	HLSW	HLSW	HLSW	HLSW	1120320	6 processors + 2 HW accelerators
4	HLHW	HLHW	HLHW	HLSW	HLSW	HLSW	HLSW	HLSW	34613	5 processors + 3 HW accelerators
5	HLHW	HLHW	HLHW	HLHW	HLSW	HLSW	HLSW	HLSW	33265	4 processors + 4 HW accelerators
6	HLHW	HLHW	HLHW	HLHW	HLHW	HLSW	HLSW	HLSW	33183	3 processors + 5 HW accelerators
7	HLHW	HLHW	HLHW	HLHW	HLHW	HLHW	HLSW	HLSW	25454	2 processors + 6 HW accelerators
8	HLHW	HLHW	HLHW	HLHW	HLHW	HLHW	HLHW	HLSW	13504	1 processor + 7 HW accelerators
9	HLHW	HLHW	HLHW	HLHW	HLHW	HLHW	HLHW	HLHW	12348	8 HW accelerators

MAC sublayer

In the case of the MAC sublayer of the HIPERLAN/2 system, the critical processes are: *access point receiver* and *access point command handler*. One of the scenarios evaluated during the architecture exploration is the reception of a Broadcast Channel PDU. According to this, a BCH PDU (which appears at the beginning of each HIPERLAN/2 frame) is being received and passed to the upper RLC sublayer. According to the HIPERLAN/2 standard, the time constraint for the specific action is 36 μs. The exploration of the various alternatives commences

by allocating both processes to software (i.e. they are characterized in OCAPI-xl as procHLSW). The simulation of the OCAPI-xl model resulted in an estimation of 25.287 cycles for executing the specific scenario. Assuming a conservative clock frequency of 50 MHz for the ARM processors, we get a time estimate of 505.74 μs which is far greater than the limit of 36 μs. As already explained in the case of the physical layer, the possibility of modeling all the processes as procManagedSW would result to even worse execution times. The simulation results illustrated in Table 4 exhibit all the alternative allocation schemes explored. A brief look reveals that alternatives (1) and (3) violate the constraint of 36 μs. Alternatives (2) and (4) result in 25.36 μs and 17.52 μs respectively for the specific scenario, which is bellow the constraint of 36 μs. Among the two, case (2) is cost efficient since it is not implemented purely in hardware, and as a result, it is the one finally selected.

Table 4. Part of architecture exploration for the MAC sublayer of HIPERLAN/2 using the OCAPI-xl approach

	receiver	Command handler	Performance in cycles	Execution Time (μs)	HW Cost
1	HLSW	HLSW	25.287	505.74	2 processors
2	HLHW	HLSW	1.268	25.36	1 processor + 1 HW accelerator
3	HLSW	HLHW	8.986	179.72	1processors+ 1 HW accelerator
4	HLHW	HLHW	876	17.52	2 HW accelerators

In the next step the high level OCAPI-xl model of the selected architecture has been refined. The refinement included the change of processes' types from high level to low level (procOCAPI1 and procANSIC). This allowed a cycle accurate simulation of the complete system functionality and confirmation that timing constraints are met.

Based on (a) the architecture exploration results, (b) the analysis of the HIPERLAN/2 computational complexity and (c) performance constraints, two core modules and two logic modules have been allocated for the realization of the HIPERLAN/2 system. Each core module includes an ARM7TDMI processor and each logic module includes a Xilinx Virtex E 2000 FPGA [11].

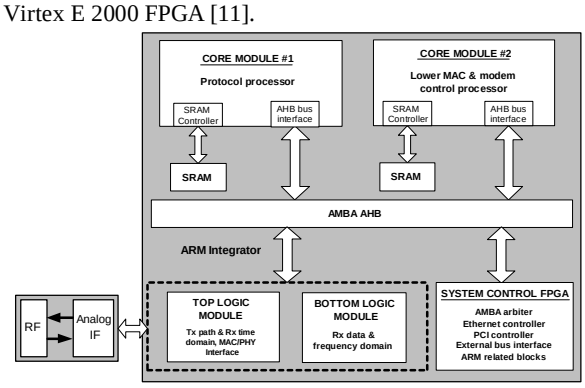

Figure 7. Architecture of selected ARM Integrator platform instance (in core module 2 change modem to physical layer)

The architecture of the ARM Integrator instance that has been selected for the realization of the HIPERLAN/2 system is shown in Figure 7. One ARM processor (indicated as protocol processor in the figure) implements convergence layer and higher DLC i.e. the functionality that was not considered during architecture exploration. The second ARM processor implements MAC sublayer functionality and physical layer control functionality. The two FPGAs implement critical parts of MAC sublayer and the digital part of the physical layer. The functionality of HIPERLAN/2 has been assigned to the allocated processing resources based on the selected assignment derived during the architecture exploration procedure presented above. Even though OCAPI-xl appears to be a promising approach for architecture exploration, there some issues that must be taken care of in order to allow OCAPI-xl's effective use in the context of real world case studies. For example, in the HIPERLAN/2 system it was difficult for the designers to create detailed models for the AMBA bus. Lack of such features could result in loss of accuracy during system model design and refinement, which in turn may lead to misleading results during the architecture exploration phase. In the case of HIPERLAN/2 system, the designers had to use external detailed models of AMBA bus which connected to the OCAPI-xl models through FLI interface (FLI stands for Foreign Language Interface and is a feature of OCAPI-xl that allows incorporation of external code into an OCAPI-xl model [9]).

5. IMPLEMENTATION RESULTS

As soon as the architecture decisions have been made, the next stage is related to the system implementation. For the tasks assigned to software processors, C++ code has been developed and mapped on the ARM7TDMI processors of the core modules. The tools used for the software development process include the Code Warrior IDE, the ARM, THUMB C and Embedded C++ compilers, the ARM.

The execution times for the basic tasks of HIPERLAN/2 DLC/MAC are presented in Table 5. The results have been obtained with an operation frequency of 50 MHz (cycle 20 ns). The code and the data for the tasks are stored in SDRAM memory. The size of the code running on the protocol processor is 1.4 Mbytes while the size of the code running on the second processor is 50 Kbytes.

Table 5. Execution times for basic tasks of HIPERLAN/2 DLC/MAC layer (where AP: Access Point, MT: Mobile Terminal, CL: Convergence Layer, Tx: Transmitter, Rx: Receiver)

MT-BCH/FCH Decoder Modem Ctrl MAC Layer tasks	Exec. Time (μs)	DLC tasks	Exec. Time
Initialisation Phase (Reset & Config @ slot commands)	1.20	AP-Scheduler	0.2 ms
Synchronisation Phase (BCH_SRCH, Rx_FCH with rpt = 1, Rx_ACH)	2.65	AP/MT-TxCL	0.6 ms
BCH decoding and BCH CRC checking	5.25	AP/MT-TxBuilder (full frame)	0.7 ms
Decoding of a single IE (UL)	3.23	AP/MT-TxBuilder Copy using DMA (580 bytes – word transfer)	1.5 μs
Decoding of 3 IEs (2 ULs, 1 DL) including CRC checking & Puncturing	1.5	AP/MT-Rx Decoder	0.4 ms
		AP/MT-RxCL	0.7 ms

For the tasks assigned to hardware, VHDL code has been developed. A typical FPGA flow has been adopted for realization of the tasks assigned on the platform's logic modules. The tools used include Modelsim (simulation), Leonardo Spectrum (synthesis), and Xilinx ISE tools (back end design). The detailed

architecture of the functionality realized by the logic modules of the prototyping platform is shown in Figure 8.

In the first FPGA (BOTTOM Logic Module) the frequency and data domain blocks of the receiver are mapped. The total utilization of the first FPGA is 85%. The second FPGA (TOP Logic Module) includes the transmitter, the time domain blocks of the receiver, the interface to MAC and a slave interface to an AMBA bus. The total utilization of the second FPGA is 89%. The utilization per resource type for the first and second FPGAs is presented in Table 6. Two clocks of 40 and 80 MHz are driven in each FPGA. The size of the configuration files for the two FPGAs is 1,2 Mbytes.

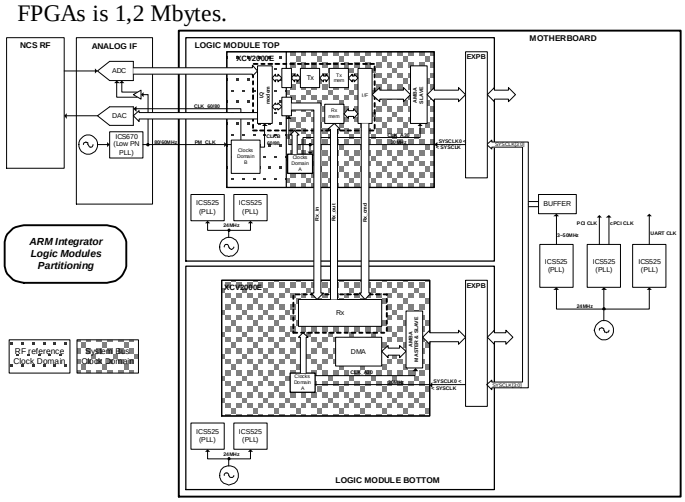

Figure 8. Architecture of the SoC on the ARM Integrator

Table 6. Utilization per resource type for the two logic modules FPGAs

Resource	Used		Utilization (%)	
	BOTTOM Logic Module	TOP Logic Module	BOTTOM Logic Module	TOP Logic Module
I/Os	93	312	18.16	60.93
Function Generators	14923	16527	38.86	43.04
CLB Slices	12164	11252	63.35	58.60
DFFs or Latches	6368	8544	15.60	20.94

The performance results presented above from the realization of the HIPERLAN/2 system on the ARM Integrator platform are expected to improve in a SoC implementation. This is due to the overheads introduced by the ARM Integrator platform architecture (FIFOs of the bus interface, SDRAM controller etc.), and also due to the lack of a local bus for the communication between the baseband processing block and the lower MAC processor (which also controls the baseband processing block). The use of the proposed system level architecture exploration approach allowed the definition of an efficient architecture that satisfied the targeted performance constraints. No time consuming iterations and feedback loops from the low level implementation stages for architecture modifications were required. This led to reduction of the design time by 30% compared to the development times of systems of similar complexities where no systematic architecture exploration was used. Additionally, the initial performance constraints set for the target system implementation have been met.

6. CONCLUSIONS

In the previous paragraphs, we have presented the process of producing an FPGA based prototype of the HIPERLAN/2 system. The prototyping process has been based on a method for efficient architecture exploration early in the design cycle, and targeted the ARM Integrator platform. The approach resulted in efficient functional debugging/verification, performance testing and validation of architecture decisions towards a SoC implementation of the system. The architecture exploration lead to correct architecture decisions early in the design cycle, and eliminatesd the need for time consuming iterations and feedback loops from the low level design stages.

7. REFERENCES

[1] IEEE Std 802.11b, Part11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Higher-Speed Physical Layer Extension in the 2.4 GHz Band, 1999.

[2] IEEE Std 802.11a, Part11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: High Speed Physical Layer in the 5 GHz Band, 1999.

[3] ARM Integrator (2005) Available: http://www.arm.com/devtools/integrator

[4] ETSI TS 101 457: Broadband Radio Access Networks (BRAN); HIPERLAN Type 2; Physical (PHY) layer.

[5] ETSI TS 101 761-1: Broadband radio access networks (BRAN); HIPERLAN Type 2; Data link control (DLC) layer; Part 1: Basic data transport functions, 2000.

[6] ETSI TS 101 761-2: Broadband Radio Access Networks (BRAN); HIPERLAN Type 2; Data Link Control (DLC) Layer; Part 2: Radio Link Control (RLC) sublayer, V1.3.1, 2002.

[7] R. van Nee, R. Prasad, OFDM for Mobile Multimedia Communications, Boston: Artech House, Dec. 1999.

[8] CoWare Inc (2005) Available: http://www.coware.com

[9] OCAPI-xl (2005) Available: http://www.imec.be /ocapi/welcome.html

[10] SystemC (2005) Available: http://www.systemc.org

[11] Xilinx: Virtex™ Data Sheet (2005) Available: http://www.origin.xilinx.com/xlnx/xweb/xil_publications_index.js.