



Project Number:	IST-026905
Project Title:	Multiple-Access Space-Time Coding Testbed
Project Coordinator:	C.F. Mecklenbräuker
Deliverable Number:	D2.3.2a

Title of Deliverable:	Report on MIMO MAC extensions
Workpackage:	WP-2
Nature:	R
Dissemination level:	PU
Editor:	Peter Lüthi
Authors:	see list inside
Contractual Date of Deliverable:	Dec. 31, 2007
Actual Date of Delivery:	Dec. 31, 2007

Abstract:

This deliverable reports on the MIMO MAC extensions for both, hardware and software. The first part outlines the hardware of the prototyping system, with a focus on the PowerPC processor and its MAC processing tasks. The second part describes the concept of the MIMO MAC and discusses two different MAC implementations on our prototyping platform.

Reporting on MAC throughput measurements and in-depth performance assessment of the implemented MU-MIMO MAC algorithms on our platform is put off to subsequent deliverables D2.3.2b and D2.3.2c due to delayed hardware extensions.

Contents

1	Platform Overview	5
1.1	FPGA Prototyping Platform	5
1.2	System Architecture	6
1.3	PowerPC Processor Subsystem	8
1.4	Hardware Integrity Tests	9
1.5	General Performance Improvements	9
1.6	Performance Numbers	10
1.6.1	Data Transfer Performance	10
1.6.2	Context Switching	12
1.6.3	Memory Latency and DMA Setup Costs	13
2	MIMO MAC Extensions	15
2.1	Implementation Considerations	15
2.2	IEEE 802.11-compliant MAC Framework	15
2.2.1	Architecture	16
2.2.2	Performance Assessment	18
2.3	MASCOT MAC	19
2.3.1	Architecture	19
2.3.2	Performance Assessment	22
2.4	Comparision	23
3	MIMO MAC Extension Summary and Roadmap	25
3.1	Summary	25
3.2	Roadmap	26

Authors

Peter Lüthi, Markus Wenk, Daniel Wagner
Eidgenössische Technische Hochschule Zürich
tel.: +41 44 632 50 08 (Peter Lüthi)
e-Mail: {luethi,mawenk,dwagner}@iis.ee.ethz.ch

Executive Summary

The aim of Task 2.3 of the MASCOT project is the development, testing, and demonstration of multi-user (MU) MIMO algorithms on a real-time FPGA-based testbed. This deliverable describes the status of this work, the implemented setup, and the planning of the work ahead. This report focuses on the MIMO MAC extensions required for being able to successfully implement and demonstrate advanced MU-MIMO MAC functionality on the testbed.

The first part of the report outlines the FPGA prototyping platform in order to provide a basic understanding of the capabilities and limitations of the underlying hardware. An architectural overview of both, the entire testbed and the PowerPC subsystem, shows the various implemented hardware blocks, the bus hierarchy, and the different bus standards used. Furthermore, selected performance numbers illustrate the raw processing and I/O performance of the embedded PowerPC CPU core and its surrounding components.

The second part describes architectural considerations and some implementation details about two different approaches of realizing a MIMO MAC on the FPGA platform. It is notable here, that the current MIMO MAC layer is designed to run entirely in software on an embedded CPU. There do not exist any components of the MAC layer in hardware so far. The first approach tried to implement a software framework for an IEEE 802.11 [2] compliant MAC. But soon, we realized that this generic, framework-based and standard-adhering approach was quite suboptimal with respect to the overall MAC processing performance and complexity. As a consequence, we decided to implement a simplified and faster MIMO MAC layer, better adapted to the needs of our CPU-centric system setup and no longer tightly adhering to the standards.

At the end of this report, a summary about the MIMO MAC extensions is given and the roadmap for future work is given.

Chapter 1

Platform Overview

In the following, we briefly outline the system architecture and the hardware components of the MIMO testbed, where the MIMO Medium Access Control (MIMO-MAC) extensions are built on. A broader overview of the various hardware components used within this testbed has already been published in an earlier project report [6].

1.1 FPGA Prototyping Platform

The main board of the entire prototyping platform is shown in Fig. 1.1, called VAMP board [6, 4]. There are two XILINX Virtex-II Pro FPGAs (XC2VP50-5-ff1517) on this board, which are interfaced with several peripheral components. The board connects to the outside world either through a 32 bit PCI bus or a Gigabit Ethernet interface. In our testbed setup, only the Gigabit Ethernet connection is used, the PCI interface remains unused. To store large amounts of data, SRAM and SDRAM memories are available on the FPGA prototyping board. SDRAM memory offers larger memory capacities, but has a slower and more complex access scheme. For our purposes, the smaller (4 MB instead of 32 MB) but faster SRAM memory has turned out to be more practical. The processor, which is running the MIMO-MAC layer, is embedded in the left XILINX Virtex-II Pro FPGA device on the VAMP board shown in Fig. 1.1. Any wireless functionality regarding radio transmission and reception is implemented on other boards, like the BAT and WING boards [6, 5]. These boards are connected to the VAMP board using the connectors high-lighted on the right side in Fig. 1.1.

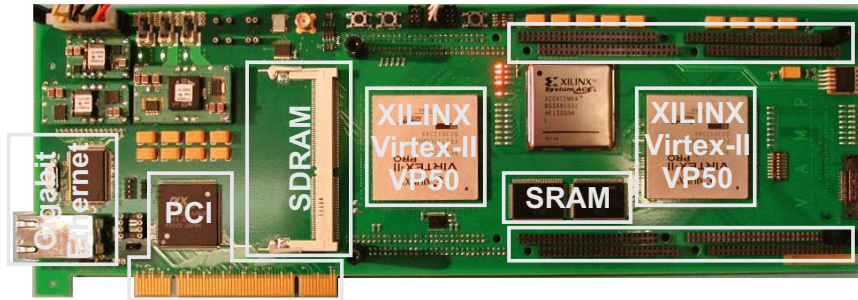


Figure 1.1: Picture of the VAMP board [4] of the ETHZ prototyping platform.

1.2 System Architecture

We first introduce the overall system architecture in order to provide the base for the understanding of the capabilities and limitations of an embedded CPU-based MIMO-MAC layer implementation.

As shown in Fig. 1.2, the entire FPGA prototyping platform consists of three major subsystems: Ethernet (ETH), MIMO, and PowerPC subsystem. The ETH subsystem is split into two parts, an external dedicated Gbit physical layer (PHY) chip on the VAMP board, and the FPGA-internal low-level MAC layer and its transmit (TX) and receive (RX) buffers. The MIMO subsystem consists as well of TX and RX buffers, and the MIMO-OFDM PHY implementation. The PowerPC subsystem incorporates the embedded IBM PowerPC 405 CPU core as hard macro, and several other blocks such as instruction and data memory, interrupt controller, and serial communication device as FPGA logic. These three subsystems are almost completely implemented on the VAMP FPGA board, only front-end parts of the MIMO PHY block and the RF circuitry reside on other boards.

The three subsystems – PowerPC, ETH, MIMO – are interconnected through a high-speed on-chip bus - the AMBA¹ *Advanced High-Performance Bus* (AHB) - for data transfer, and a low-speed on-chip bus - the AMBA *Advanced Peripheral Bus* (APB) - for configuration purposes and status information. Any payload data are solely transferred using the AHB high-speed bus. Data transfers on the AHB bus can only be initiated using AHB *master* devices, depicted as capital letter *M* in Fig. 1.2, and must target an AHB *slave* interface, depicted as capital letter *S*. In our system, AHB master devices are represented by the PowerPC processor using its PLB2AHB bridge, and the *Direct Memory Access* (DMA) engines of the ETH and MIMO sub-

¹AMBA: Advanced Microcontroller Bus Architecture

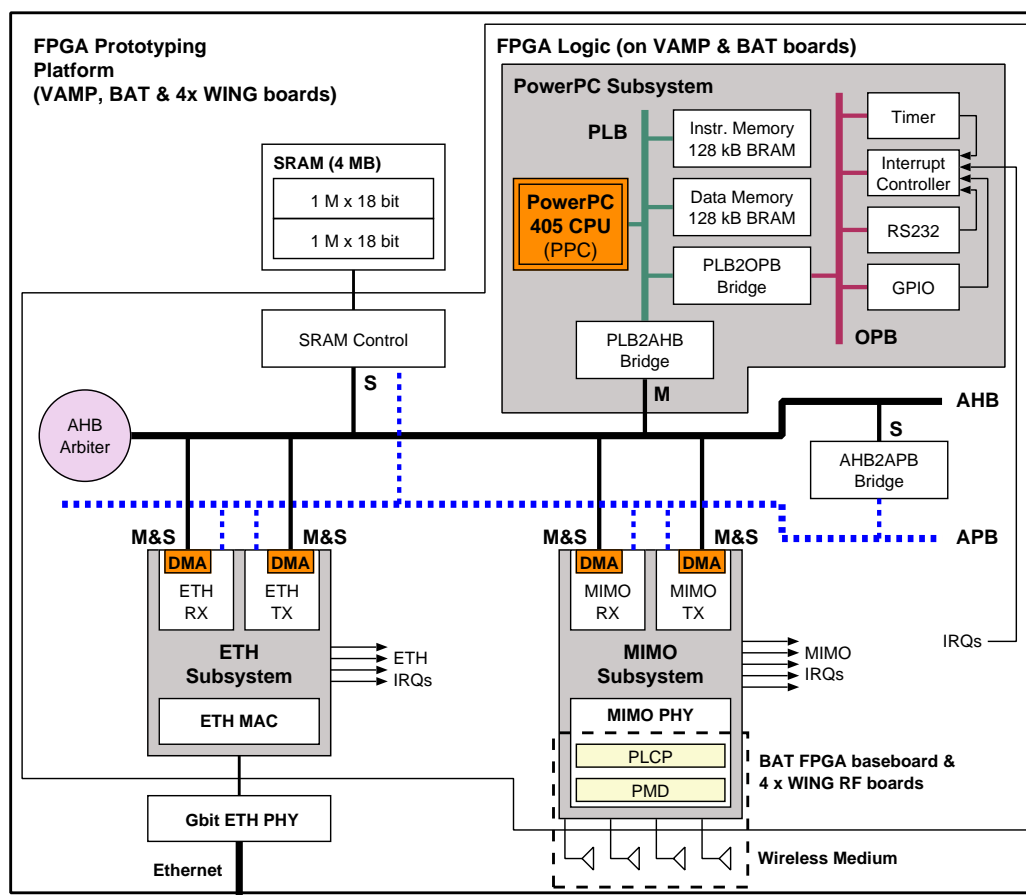


Figure 1.2: Block diagram of the entire MIMO-OFDM FPGA prototyping platform consisting of three major subsystems: Ethernet (ETH), MIMO, and PowerPC subsystem.

systems. AHB slave devices are represented by the SRAM, the transmit (TX) and receive (RX) buffers of the ETH and MIMO subsystems, and the AHB2APB bridge. The low-speed APB bus serves as peripheral bus for configuration purposes, e.g. the setup of the DMA engines, or for retrieving status information from the ETH and MIMO subsystems.

Most of the FPGA logic in our prototyping system is clocked at 80 MHz, the system clock frequency. The system clock is distributed across all FPGAs and comes from a single 80 MHz clock source on the BAT board. Any other clocks with different frequencies, for instance the PowerPC clock, are derived from the 80 MHz system clock in order to ensure a purely synchronous system.

1.3 PowerPC Processor Subsystem

The MIMO MAC layer is mainly implemented in software on the embedded IBM PowerPC 405 processor in one of the two XILINX Virtex-II Pro FPGAs on the VAMP board. The PowerPC 405 CPU core itself is implemented as hard macro on the FPGA die and is able to run up to clock frequencies of 350 MHz, depending on the speed grade of the FPGA device. In our system, the PowerPC CPU runs at a speed of 240 MHz. This clock frequency is in accordance with the FPGA device speed grade, and it is also an integer multiple of the overall system frequency of 80 MHz.

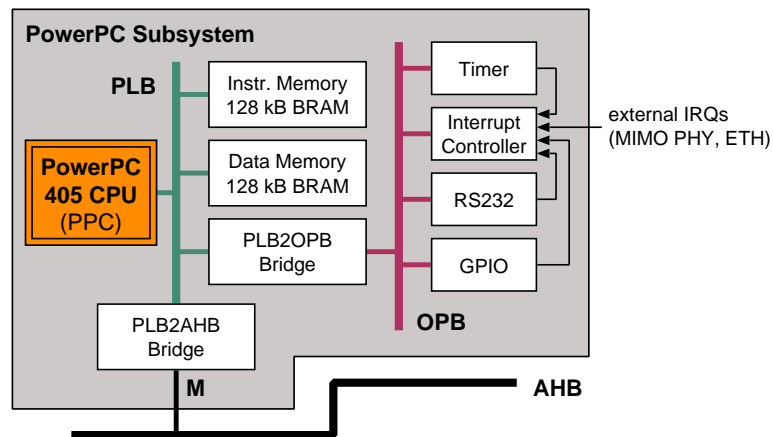


Figure 1.3: Block diagram of the PowerPC subsystem

The PowerPC 405 CPU is part of the PowerPC subsystem, as shown in Fig. 1.3. The PowerPC processor itself consists of the CPU core and an integrated cache controller for both, instruction and data. The cache controller is directly connected to the *Processor Local Bus* (PLB) and fetches instruction and data from two dedicated block memories, also attached to the PLB bus. Any additional processor infrastructure such as interrupt controller, programmable interrupt timer, serial communication, or general purpose I/O (GPIO) are available as synthesizable FPGA IP blocks from XILINX and located at the *On-Chip Peripheral Bus* (OPB). The PowerPC can access the OPB devices through a bridge connecting PLB and OPB. Since major parts of the FPGA prototyping system are connected to the AHB bus, the PowerPC subsystem incorporates also a PLB2AHB bridge to provide AHB connectivity for the CPU.

A suboptimal characteristic of the PowerPC CPU architecture with respect to embedded systems is the demand for a large front-side bus. The native PowerPC CPU has a 128 bit front-side bus, the embedded variant

still asks for 64 bit. This has led to a PLB bus implementation from XILINX using a data width of 64 bit. XILINX admits that this implementation has room for improvements, especially for the Virtex-II Pro FPGAs. According to XILINX, upcoming implementations of the PLB IP will address this issue and some improvements are promised.

1.4 Hardware Integrity Tests

To ensure proper operation of the entire prototyping system, three different loopback tests were implemented in the PowerPC in order to provide a step-by-step verification procedure to check the integrity of the hardware at different levels:

1. A higher-level, PowerPC-centric MAC loopback test checks the basic operation of the PowerPC subsystem including the AHB devices and the external SRAM – without checking any MIMO-PHY components. Only the AHB-attached transmit and receive buffers of the MIMO-PHY are involved in this loopback test: They are internally directly connected together in order to form the required loopback for the test. Furthermore, an extensive memory read and write test is carried out on all memory blocks including the RX and TX buffers in the ETH and MIMO subsystems. The memory test involves several data transfer operations such as read and write operations for byte, half-word, and word size, as well as cache-enabled and cache-disabled data transfer patterns.
2. A lower-level PHY loopback test stimulated from the PowerPC and targeting the MIMO-PHY ensures the integrity of the MIMO-PHY layer except for the operations taking place on the BAT board.
3. An extended low-level BAT loopback test – stimulated from the PowerPC as well – checks the integrity of the entire digital system including up- and downsampling blocks in the BAT FPGA right in front of the RF circuitry at the WING boards.

1.5 General Performance Improvements

In order to improve the MAC performance, i.e., the end-user data throughput, the following system parameters have been subsequently improved:

- We were finally able to improve the PowerPC operating frequency from 160 MHz to 240 MHz. This is the highest achievable PowerPC operating frequency for our FPGA prototyping system. The CPU operating frequency is ultimately limited by the FPGA device speed grade. The increase of the CPU clock leads to a speed up of 50% for the CPU performance – as long as data and instructions are read-out from the CPU cache. Since the PLB bus frequency of 80 MHz can not be increased easily, the cache miss penalty remains the same as before.
- The size of the PowerPC-centric instruction and data memories has been increased from a total of 128 kB to 256 kB on-chip memory. 256 kB constitutes the maximum size of directly attachable PowerPC memory, due to vendor-specific limitations in the implementation of the processor local bus and memory controller.
- As CPU-based APB register access has emerged being a very costly operation in terms of wasted CPU clock cycles, the number of individual APB register accesses has been consequently reduced by an improved utilization of the configuration space, i.e., less individual configuration locations. As an example, only two APB registers need to be configured for the ETH DMA setup, and only four registers need to be programmed for the MIMO DMA setup.

1.6 Performance Numbers

1.6.1 Data Transfer Performance

The data transfer performance is a key factor for the overall system performance in environments involving a lot of data exchange. In our system, data transfers occur either between the CPU and a peripheral block, or between two peripheral blocks. These data transfers are usually block-oriented and move data from one memory to another, either initiated by the CPU itself or by a DMA engine. Therefore, we can split all kinds of data transfers into two different classes: CPU-based and DMA-based data transfers.

The CPU-based data transfer performance of the PowerPC processor is compared with the DMA-based data transfer performance in Fig. 1.4 and Fig. 1.5 for small and large block sizes, respectively. The CPU-based data transfers represent a data copy from one peripheral memory to another peripheral memory, carried out solely by the CPU. The DMA-based data transfers represent as well a data copy from one peripheral memory to another

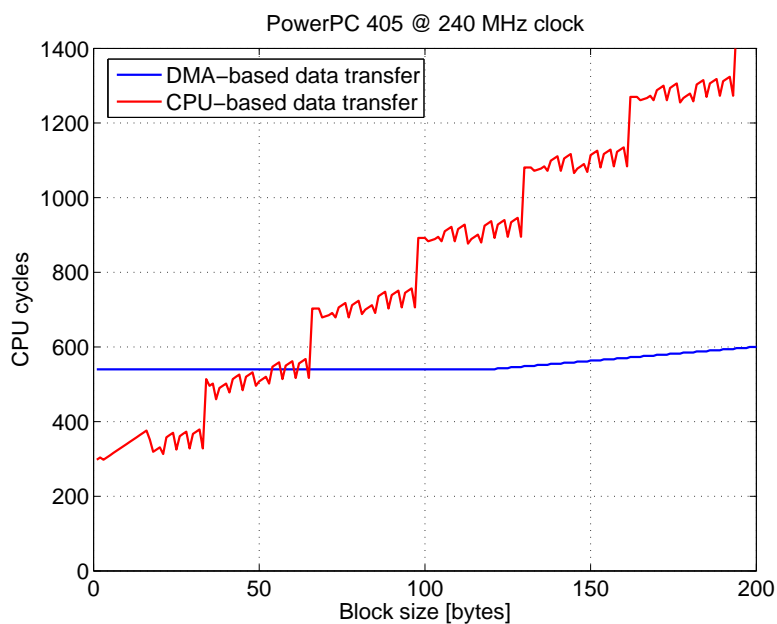


Figure 1.4: Comparison of data transfer rates: CPU-based and DMA-based data transfer times for small block sizes.

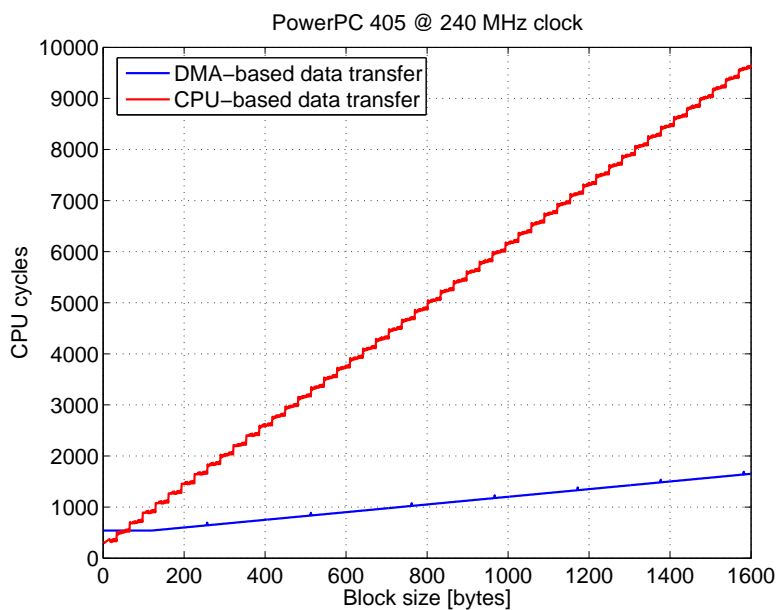


Figure 1.5: Comparison of data transfer rates: CPU-based and DMA-based data transfer times for large block sizes.

one, but also include the required DMA setup procedure to be carried out by the CPU.

As illustrated in Fig. 1.4, CPU-based data transfers are only more effective in terms of CPU cycles than DMA-based ones, if the block size for the data transfer stays below 60 bytes. It is notable here, that all data transfers originating from the Ethernet block have a minimum size of 60 bytes². At the MIMO side, frames shorter than 60 bytes occur only if no payload data is involved, i.e., in case of native control or management frames.

1.6.2 Context Switching

Another interesting parameter for the assessment of the overall system performance is the context switching overhead for the CPU. Context switching is necessary right before and after the execution of an interrupt service routine (ISR). Whenever an interrupt (IRQ) is flagged to the CPU, the CPU needs to immediately save the current context (i.e., the contents of CPU registers and flags) to the stack in order to carry out the code of the corresponding ISR. The effort for this context switch depends on the CPU architecture and the compiler-specific conventions. The context switch before the execution of the ISR is called *prologue*, the context switch after the ISR execution is called *epilogue*.

The overall interrupt latency consists of the following parts: Initially, the time elapsed from asserting the interrupt signal at the peripheral device until this is recognized and handled by the interrupt controller. Once the IRQ controller has solved the origin, prioritization and masking of the IRQ event, it flags a non-critical external IRQ to the CPU. The CPU then starts with the prologue, saves the current context, reads out the status registers of the IRQ controller and then decides, which ISR it has to launch in order to service the pending interrupt. The prologue ends when the CPU starts executing the corresponding ISR code. These actions sum up to a certain processing delay or IRQ latency, and thus form the limiting factor for the interrupt response time.

Measurements for both, the prologue and epilogue³ showed total CPU costs for the context switch of approximately 420 CPU cycles, as listed in Tbl. 1.1. For the CPU running at 240 MHz, this amounts to 1.75 μ s. For the entire system, having for instance 2000 times per second serviced all

²Ethernet frames being shorter than 60 bytes will be padded to 60 bytes before being physically transmitted across the Ethernet wire.

³The current low level implementations of the prologue and epilogue are part of the XILINX tool chain. Although they are not completely optimized for our use case, the provided implementations still deliver sufficient performance.

currently used interrupts (mainly 9 different sources), this sums up to approximately 3.2% of the CPU performance being spent purely on context switching activities, leaving a lot of CPU power for actual computing tasks.

Table 1.1: CPU costs for ISR context switching.

ISR prologue	approx. 300 CPU cycles
ISR epilogue	approx. 120 CPU cycles
Total ISR context switch	approx. 420 CPU cycles

1.6.3 Memory Latency and DMA Setup Costs

The various latencies incurred by accessing different memories from the CPU are summarized in Tbl. 1.2 and Tbl. 1.3. In our system, we have caching enabled and only perform transfers of (payload) data on cacheable – and thus prefetchable – memory locations. CPU-based access on cacheable memory locations are executed using cacheline burst transfers. Only transactions involving device configuration or status read-out are targeting uncached – and thus non-prefetchable – address space. These transactions are carried out using single transfers, for instance transactions targeting the APB address space. The APB transactions are quite costly, not in terms of bandwidth but latency. Increasing the CPU frequency does not help to speed up those accesses since the CPU is most of the time waiting for the external hardware to complete the transaction. For the configuration of the DMA engines in the MIMO subsystem, four APB accesses have to be made, for setting up the ETH DMA engines, only two APB transactions are necessary. This corresponds to an effective DMA setup time of approximately 1.0 μ s and 0.5 μ s for MIMO and ETH subsystem, respectively.

Table 1.2: CPU-based memory read latency for single word transfers.

Read word, DCache hit	3 CPU cycles
Read word, DCache miss, from BRAM	42 CPU cycles
Read word, DCache disabled, from BRAM	32 CPU cycles
Read word, DCache miss, from SRAM	63 CPU cycles
Read word, DCache disabled, from SRAM	41 CPU cycles
Read word from APB register (no caching)	68 CPU cycles

The CPU costs for configuring the DMA engines are listed in Tbl. 1.4. The setup costs vary depending on the number of APB registers required for

Table 1.3: CPU-based memory write latency for single word transfers.

Store word, DCache hit	3 CPU cycles
Store word, DCache miss, to BRAM	42 CPU cycles
Store word, DCache disabled, to BRAM	32 CPU cycles
Store word, DCache miss, to SRAM	63 CPU cycles
Store word, DCache disabled, to SRAM	29 CPU cycles
Store word to APB register (no caching)	56 CPU cycles

configuration. While the theoretical DMA bandwidth is identical to the AHB bus bandwidth, it only makes sense to assess practical throughput figures based on dedicated, realistic block sizes. For a common Ethernet frame size of 1518 bytes, the achievable DMA throughput is limited to 1.87 Gbit/s.

Table 1.4: Performance and CPU costs for DMA transfers.

DMA setup costs	120 - 240 CPU cycles
DMA bandwidth	4 bytes per 3 CPU cycles = 2.56 Gbit/s (i.e. one AHB data transfer cycle)
DMA transfer time ⁴ [1518 bytes]	approx. 7.0 μ s
DMA throughput ⁴ [1518 bytes]	approx. 1.87 Gbit/s

⁴ These values represent the view point of the CPU, i.e, the values include the hardware and software overhead for interrupt signalling and context switching. In our system, the DMA throughput based on bus occupancy only reaches up to 2.53 Gbit/s for 1518 bytes.

Chapter 2

MIMO MAC Extensions

2.1 Implementation Considerations

An important criterion for the design and implementation of a MAC protocol for our MU-MIMO prototyping system was the compliance to existing standards such as the industry-wide adopted IEEE 802.11-2007 [2] or the upcoming IEEE 802.11n/D2.0 [3] standards.

First, we tried to implement a MU-MIMO MAC based on a software framework adhering as close as possible to the existing standards. But soon, we realized that this generic, purely software-based approach had detrimental effects on the overall MAC processing performance. As a consequence, we decided to implement a simplified and faster MIMO MAC layer, better adapted to the needs of our CPU-centric system setup and no longer tightly adhering to the standards. Seeking to comply with existing industry standards was no longer an option for the MIMO testbed employing a general purpose CPU. We rather tried to adapt and optimize specific tasks of the MIMO MAC to the PowerPC architecture and the testbed infrastructure.

In the following, an IEEE 802.11-compliant MAC framework and a simplified MAC architecture are described and compared.

2.2 IEEE 802.11-compliant MAC Framework

During a master thesis [7], the development of a software framework for a IEEE 802.11-compliant MAC has been addressed. The MAC architecture was specified and designed based on the IEEE 802.11-2007 [2] standard, and laid out to integrate the IEEE 802.11n/D2.0 [3] standard in the future as well. A bring-up version of the software-based MAC architecture has been implemented in the testbed. It was designed for a system setup consisting of

one *Access Point* (AP) and several *Stations* (STA). The system employed a simple *Point-Coordination Function* (PCF) running on the AP for communicating and exchanging data with the STAs. The centralized coordination method in the AP used to poll the STAs periodically.

Although this software framework has been specified and implemented carefully whilst adhering rather strictly to the standards, the first implementation results showed quite disappointing results with respect to the MAC processing performance. One of the main reasons for the rather bad performance was the modular and generic structure of the software, seeking to implement all control structures in a well-ordered hierarchy, but with detrimental effects on the overall processing performance.

The software framework has not been completely functionally verified on the FPGA prototyping platform, since parts of the hardware were not ready at that time. Therefore, no extensive testing of the MAC framework has been carried out. Nevertheless, the assessment of the software architecture in order to get some core performance values was still possible.

2.2.1 Architecture

The graphical description of the software architecture is provided in Fig. 2.1. It shows the control flow of an IEEE 802.11-compliant MAC architecture. The large rectangles containing one or more smaller beveled rectangles are called *blocks*. The smaller beveled rectangles represent *processes* inside a block.

A *block* is a logical top-level unit encapsulating different structures such as processes and interrupt handlers. It only appears inside the documentation to encapsulate logically related components.

A *process* forms a functional unit for a certain task and contains a finite state machine to keep track of the current working state. Each process is repeatedly triggered, either periodically or event-driven.

The process scheduling algorithm – not to be confused with the PCF-based MAC scheduling algorithm – ensures that all processes are executed properly and timely. The implemented process scheduling algorithm is based on a simple non-preemptive round-robin scheme – also called cooperative scheduling. The decision to take this very simple scheduling method was based on several criterias such as fairness and implementation complexity. Last but not least, it was also too early in the implementation stage for applying timing restrictions and latency optimizations.

The control flow of the MAC architecture depicted in Fig. 2.1 is based on the original proposal presented in Appendix C of the IEEE 802.11-1999 [1] standard, but described in a more general way. The aim thereby was to

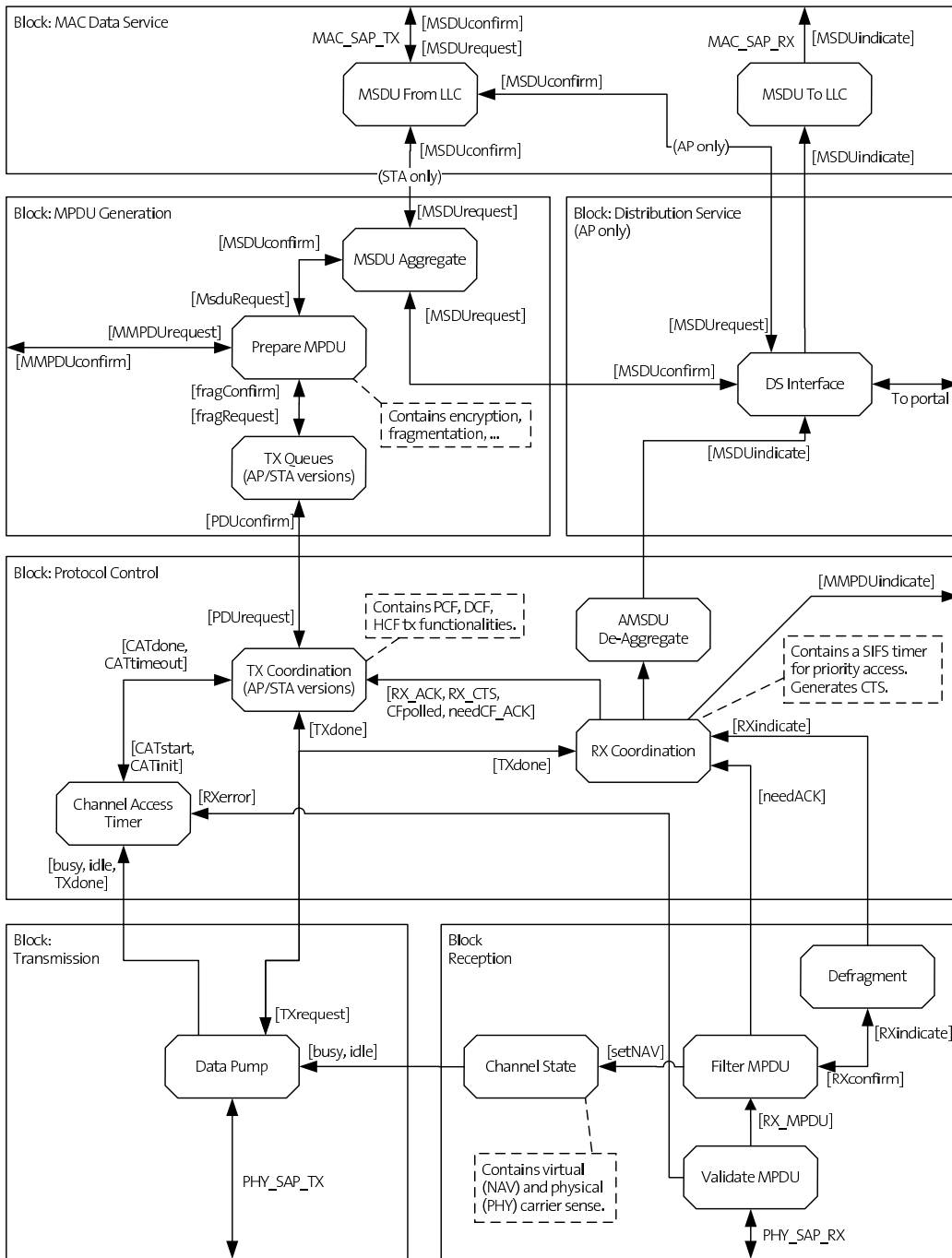


Figure 2.1: Layered software architecture showing the control flow of a IEEE 802.11-compliant MAC.

obtain an architecture being able to also integrate the IEEE 802.11n/D2.0 [3] standard.

At the top of the figure, the *MAC Data Service* block is shown, implementing the interface to the *Logical Link Control* (LLC) layer. The processes *MSDU¹ From LLC* and *MSDU To LLC* realize the MAC service access point (SAP) interface as specified in IEEE 802.11.

Below the MAC Data Service block, two other blocks called *MPDU² Generation* and *Distribution Service* (DS) are located. The DS block only exists in APs and globally decides where to forward packets³ to. Packets can be sent from the DS either upwards to the LLC layer, to the right using the *Portal* in order to employ a different IEEE 802-based MAC layer (such as IEEE 802.3 Ethernet), or to the left for wireless transmission from this MAC layer.

The *MPDU Generation* block creates and prepares the MPDU frames for transmission over the wireless interface. MPDU frame generation covers several aspects such as data encryption, fragmentation, and packet queueing. In case of STAs, this block directly interacts with the MAC Data Service block, otherwise the Distribution Service block is in between those.

The next large block, called *Protocol Control*, includes the different channel access schemes, namely DCF, PCF and HCF as well as PS-Poll response, acknowledgment (ACK), RTS/CTS protection, and different retry mechanisms. The process *TX Coordination* handles the protocol rules on the transmit side, whilst process *RX Coordination* reacts on frame reception events.

The last two blocks, *Transmission* and *Reception*, perform low-level tasks. The process *Data Pump* realizes a queue for immediate frame transmission, whenever a process inside the Protocol Control block flags a transmission start event. The Reception block carries out frame validation, filtering and de-fragmentation, and also observes the activity on the wireless medium (physical carrier sense).

2.2.2 Performance Assessment

The IEEE 802.11n/D2.0 [3] standard states a maximum MAC processing delay of 2 μ s and a maximum PHY layer delay of 14 μ s, summing up to the required 16 μ s timing for the *Short Interframe Space* (SIFS). Since the MIMO

¹MSDU: MAC Service Data Unit, i.e., frames at top-level MAC layer

²MPDU: MAC Physical Data Unit, i.e., frames at bottom-level MAC layer

³In the following, the term *packet* is used as a synonym for the word *frame*, representing some kind of structured data.

PHY layer in our testbed does not meet the specified PHY layer delay, we solely concentrate on the MAC processing delay within this analysis.

At the time, when the performance assessment of this MAC implementation took place, a PowerPC CPU clocked at 160 MHz was used, since the latest hardware performance improvements for the 240 MHz CPU clock were not yet available.

The performance measurements carried out on this MIMO MAC framework have shown that between 200 μ s and 300 μ s are necessary for the complete transmission of a short frame. This time was measured from the point where the process *TX Coordination* starts until the *TX Done* interrupt signals the end of the transmission. Similar results were obtained for the receive chain: The measurement started at the interrupt originating from the reception of a new frame and stopped at the completion of the process *RX Coordination*. The cause for these rather large processing delays was found to be accumulated mainly in the rather generic and large software framework not being optimized for the special needs of the embedded CPU and testbed architecture.

2.3 MASCOT MAC

The MASCOT MAC represents a simplified MAC architecture streamlined for the needs of our MIMO testbed. It emerged as a result of previous work about MAC processing on our testbed, such as the approach described in Sec. 2.2.

2.3.1 Architecture

The MASCOT MAC employs a simple *Point-Coordination Function* (PCF) to serve all clients. This centralized coordination scheme was chosen in order to avoid any complex scheduling algorithms or sophisticated hardware-based carrier-sense mechanisms. Moreover, the PCF-based approach allows for good controllability of the entire test setup, since only one master – the AP – is present in the system. Last but not least, the PCF protocol allows only one device, AP or STA, to transmit data, and thus avoids frame collisions completely. The currently implemented scheduling algorithm is based on a simple round-robin scheme, providing the robustness and fairness required for a complex system bring-up. The scheduler polls each STA to retrieve information about pending payload to be transferred. The scheduler does not prioritize any STA, although this could be beneficial for certain application scenarios. The prioritization scheme can then be implemented based on the

STA's transmit queue fill level or the traffic class.

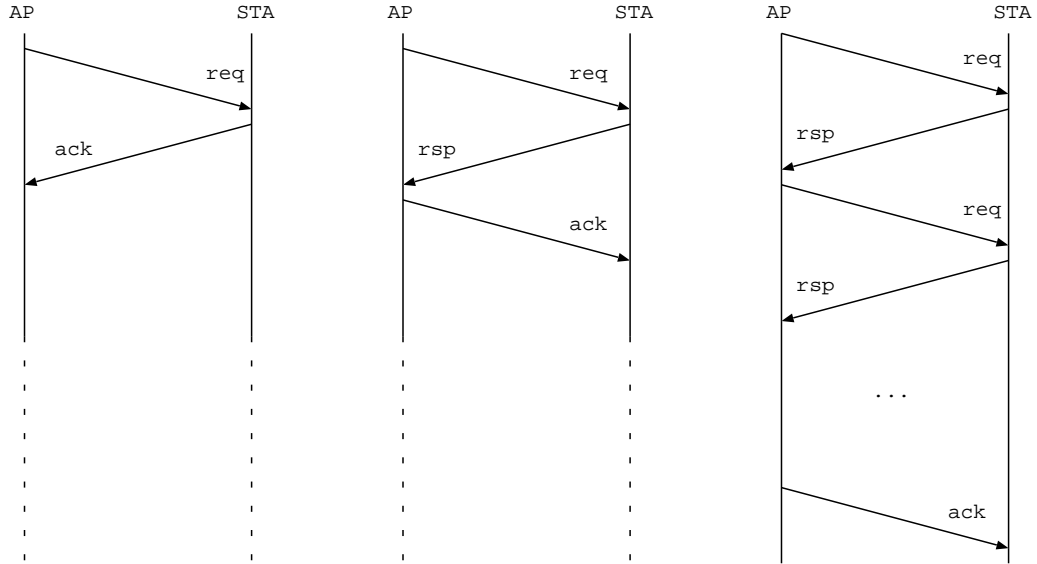


Figure 2.2: Low level communication protocol of the MASCOT MAC.

The low level communication protocol between AP and STAs for the MASCOT MAC is shown in Fig. 2.2. Only the AP is allowed to initiate a sequence of transactions. This means that no STA is allowed to start a transaction on its own. The AP initiates a sequence of transactions by transmitting a request packet⁴ (REQ) to a specific STA. The addressed STA is allowed to send exactly one packet back to the AP. This packet can either be an acknowledge (ACK) packet without payload or a response (RSP) packet containing payload. If the STA has returned an ACK packet (contains no payload), the AP is free to either send again payload to this STA, or to service the next STAs. In case of a returned RSP packet (contains payload), the AP needs to return either an ACK packet to terminate the sequence of transactions, or to send another REQ packet containing payload.

All kind of packets contain meta information in the header section for the MAC processing. All devices, AP and STAs, have a unique identifier (ID) to allow for dedicated device addressing. At the moment, the ID has to be configured by hand at every system start.

Automatic discovery of new devices is not implemented in the MASCOT MAC since the benefit thereof is only small, but would increase the complexity of the entire system. The AP simply knows from the beginning, how

⁴Again, the term *packet* is used as a synonym for the word *frame*.

many STAs are present in the setup. Nevertheless, the automatic discovery feature can still be added later.

A high-level block diagram of the MASCOT MAC is shown in Fig. 2.3. The MASCOT MAC is divided into two layers. The lower part handles the low-level protocol including all services and communication for the hardware blocks. It is completely interrupt driven. If a hardware device is configured to be an AP, this layer constantly polls the STAs. The upper part consisting of the *Packet Scheduler* constitutes the high-level protocol. It defines the sequence of polling and incorporates more sophisticated parts of the MAC algorithm. The purpose of this layered scheme is to abstract MAC-specific algorithmic parts from the lower-level hardware-centric processes.

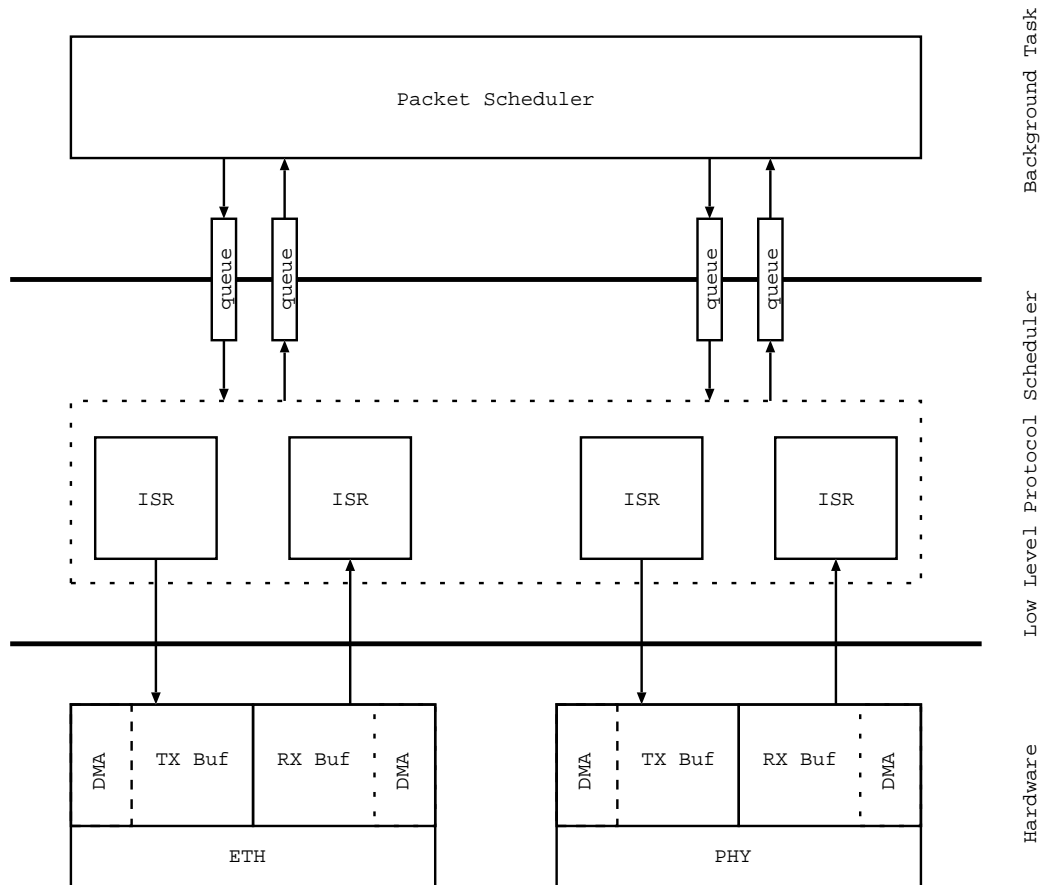


Figure 2.3: Overview of the MASCOT MAC architecture

Header Information

Every packet contains a meta data for the MAC processing. The meta data is collected in the *MAC Header* at the beginning of each packet. The MAC header has a size of 32 bytes – exactly the size of a PowerPC cacheline – and is always cacheline-aligned. This characteristic simplifies the implementation of the MAC software considerably and ensures a consistent handling of the MAC header in our system. (The subsequent payload is cacheline-aligned as well, because this is a requirement of the DMA engines.) The MAC header consists of the following items, as depicted in Fig. 2.4:

- **size**: size of the complete packet [bytes]
- **src**: source address ID
- **dst**: destination address ID
- **pkt_id**: unique packet ID (for debugging purposes)
- **pkt_type**: type of packet (REQ, RSP, ACK)
- **pld_type**: type of data payload (EMPTY, DATA, PING)
- **pld_seq**: sequence number of transmitted payload
- **ack_seq**: sequence number of acknowledged payload
- **sta_feedback**: feedback from station (currently unused)

2.3.2 Performance Assessment

The performance assessment for the MASCOT MAC was carried out using a PowerPC CPU running at a frequency of 240 MHz. Only initial performance measurements have been carried out, but they showed quite promising results.

The response time of the MAC is a critical factor for the overall system throughput. The response time of the MASCOT MAC is defined by the time elapsed from the start of the PHY RX ISR until the MIMO PHY hardware is instructed to transmit the next frame. The shorter this time, the better the responsiveness of the MAC, the more frames can be transmitted within a given time interval.

The processing delays for the current MASCOT MAC implementation are listed in Tbl. 2.1, assuming in-advance allocated transmit data. The numbers are different for AP and STA, since the AP needs to perform more complex processing tasks than the STA.

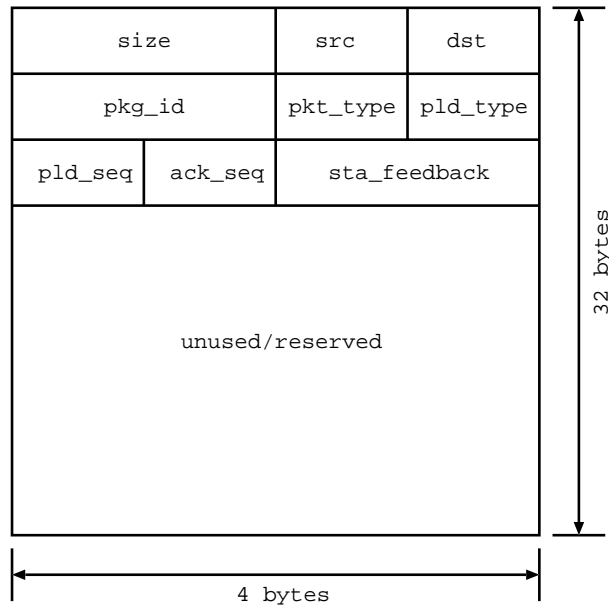


Figure 2.4: Structure of the MASCOT MAC header

Table 2.1: Frame-size independent MAC processing delay, assuming in-advance allocated transmit data.

MAC processing delay: Time elapsed from entrance of MIMO RX ISR until MIMO PHY hardware is instructed to transmit the next frame.	
AP MAC processing delay	3.4 μ s
STA MAC processing delay	2.4 μ s

2.4 Comparison

The results of the MASCOT MAC described in Sec. 2.3 clearly outperform the ones of the 802.11-compliant MAC framework described in Sec. 2.2. Although we did not carry out complete MAC-level throughput assessments, the MAC processing delay – reflecting the responsiveness of the MAC layer – is a valid performance indicator to compare both MAC implementations.

Already in early stages of the development of the IEEE 802.11-compliant MAC framework, the drawbacks of the generic approach were recognizable. Moreover, the huge engineering effort to implement and verify a standard-compliant MAC layer compared to the additional, minor benefits for our testbed do not justify this approach.

The approach of the MASCOT MAC goes into another direction. It does no longer strictly adhere to the IEEE 802.11n/D2.0 standard and thus

neglects any performance-detrimental implications of the standard for our hardware and software architecture. The MASCOT MAC also takes the capabilities and limitations of our hardware into account.

Chapter 3

MIMO MAC Extension Summary and Roadmap

3.1 Summary

In the risk analysis in the previous deliverable D2.3.1 [6], three main risks were listed for hindering a timely delivery of milestone M2.3.2:

- Delays in the redesign of the VAMP2 platform.
- Underestimation of the effort required for the implementation of a MU-MIMO demonstration.
- Lack of processing resources for successfully implementing a MU-MIMO MAC layer on the hardware platform.

Unfortunately, two of the risks listed above have become real. First, the outsourcing of the redesign of the *VAMP2* board to an experienced company finally showed excessive costs due to an increased design complexity of the new board. While the first offer of the company was within the budget, the final offer after profound complexity estimation was significantly higher and no longer within the budget. Therefore, it has been decided to stay with the system architecture of the current VAMP board – commonly referred to as *VAMP1*. Nevertheless, staying with the current board architecture still called for a partial redesign of the FPGA board – in the following referred to as *VAMP1b*), mainly because of already faded-out electronic components such as the XILINX PROM device. The VAMP1b design has been successfully sent out for manufacturing and assembling, and returned shortly. So far, the replication of the hardware for the prototyping platform using the new VAMP1b design looks promising.

Second, the overall effort required for leveraging all the different hardware components in order to establish the entire MIMO prototyping platform including analog and digital parts turned out to be more demanding than expected. The changes and fixes in the existing design, especially the required adaptations for interfacing the various blocks correctly across clock and FPGA boundaries, posed numerous design and verification challenges. Because of the complexity of the entire system, it is of utmost importance to provide a rather flawless operation of the underlying hardware prior to advancing to the implementation of sophisticated software algorithms.

Despite the tackling of all the hardware-related issues, the implementation of a software-based MU-MIMO MAC should be feasible. Of course, initial performance assessment of basic MAC functionality showed challenges in terms of meeting latency and throughput requirements comparable to IEEE 802.11n: Using a general-purpose CPU - such as the embedded PowerPC core in the XILINX FPGAs - allows for implementing basic MAC-functionality, but might not be sufficient for satisfying IEEE 802.11n comparable throughput requirements. As initial step, a greatly simplified MAC protocol is being implemented in order to assess real-world and real-time aspects of the entire prototyping system. Note that our focus resides primarily on assessing various aspects of the overall system, and not on reaching maximum throughput figures. Of course, we give best effort in attaining a high throughput on our system.

3.2 Roadmap

The current ETHZ activities are all focused on the real-time testbed demonstration at the first ETHZ open house event in February 2008. Strategic decisions on how to proceed with the implementation and assessment of new advanced MU-MIMO MAC algorithms have not yet been made. However, an ETHZ-internal meeting for discussion of further MU-MIMO activities on the testbed is scheduled for January 2008. We will discuss the future focus of the testbed project and get an alignment with respect to current testbed capabilities and future requirements. This will provide the base for discussions about the evaluation and implementation of candidate MU-MIMO MAC algorithms suitable for the testbed. The earliest possible starting point for implementation activities concerning advanced MU-MIMO MAC algorithms will be March 2008.

A summary of upcoming deliverables for MASCOT WP 2.3 is given in Tbl. 3.1. Deliverable D2.3.2a constitutes the first issue in a series of three reports. Deliverable D2.3.2b will focus on testing of implemented MIMO MAC

algorithms, while deliverable D2.3.2c will report about the overall results for the second ETHZ open house event.

Table 3.1: List of upcoming deliverables for MASCOT WP 2.3

Task	Subject	Deadline
Deliverable D2.3.2a	MIMO MAC extensions	M24
First ETHZ open house event	testbed demonstration	M26
Deliverable D2.3.2b	testing of MAC algorithms	M30
Deliverable D2.3.2c	overall results	M36
Second ETHZ open house event	testbed demonstration	M36

Bibliography

- [1] ANSI/IEEE 802.11-1999. *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 1999 edition. Reference number: ISO/IEC 8802-11:1999. [16](#)
- [2] ANSI/IEEE 802.11-2007. *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 2007 edition. Reference number: IEEE Std 802.11-2007. [4](#), [15](#)
- [3] ANSI/IEEE 802.11n-2007/D2.0. *Draft STANDARD for Information Technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Enhancements for Higher Throughput*, 2007 edition. Reference number: IEEE P802.11n/D2.00. [15](#), [18](#)
- [4] J. Biveroni and R. Bischoff. Generic Hardware Prototyping Platform, Apr. 2004 – Sep. 2004. Supervisors: A. Burg, S. Haene, and D. Perels. [5](#), [6](#)
- [5] F. Buehler and M. Casty. MIMO RF Frontend, Apr. 2006 – Sep. 2006. Supervisors: D. Perels and D. Baum. [5](#)
- [6] A. Burg, P. Luethi, and M. Wenk. Deliverable D2.3.1: Hardware extension and MAC upgrade for MIMO testbed, Apr. 30 2007. IST-026905 MASCOT. [5](#), [25](#)
- [7] M. Burri. IEEE 802.11 Multi-User MAC framework, Feb. 2007 – Aug. 2007. Supervisors: M. Lerjen, P. Lüthi and D. Baum. [15](#)