**Exzellenzcluster**
**Cognitive Interaction Technology**

Kognitronik und Sensorik
Prof. Dr.-Ing. U. Rückert

# Application-driven Exploration of a Programmable Platform for Wireless LAN

zur Erlangung des akademischen Grades eines

Doktor-Ingenieur (Dr.-Ing.)

der Techischen Fakultät
der Universität Bielefeld

genehmigte Dissertation

von

Dipl.-Inf. Hans-Peter Loeb

| | |
|---|---|
| Referent: | Prof. Dr.-Ing. Ulrich Rückert, Universität Bielefeld |
| Koreferent: | Prof. Luciano Lavagno, Ph.D., Politecnico di Torino |

Tag der mündlichen Prüfung: 2. Februar 2012

München im August 2011
DISS KS / 04

# Abstract

Medium Access Control (MAC) plays a central role in communication devices for shared-medium protocols commonly found in home networks such as Wireless LAN (WLAN). With ongoing protocol evolution, this MAC layer has grown and still grows in complexity and its requirements (e.g., real-time). It now poses a unique challenge to a system's design. But adaptability as required for such evolving and heterogeneous protocols cannot be provided by today's conventional, dedicated devices. New and disciplined approaches to the development of flexible platforms are thus required that allow for trading off architectural features in the light of tough market competition.

Consequently, we propose a novel universal MAC concept that promotes scalability, flexibility, and ease-of-use. It is complemented by an application-driven methodology and framework for productively modeling, exploring, and implementing efficient systems. This framework is extended to address the complexity and the real-time requirements of MAC protocols in a productive way. A fully-functional and executable system model of IEEE 802.11n Wireless LAN is presented as an example. The model drives architectural design space exploration, resulting in indicative system cost estimates and a platform architecture template. A software-based FPGA prototype is implemented that demonstrates architectural concepts and re-uses the system model as device firmware. The resulting protocol-agnostic system is a competitive platform for MAC systems that proves the capabilities of our comprehensive development flow.

# Acknowledgement

I would like to take this opportunity and thank everyone who supported me during the ups and downs of writing this dissertation.

Christian Sauer gave me the chance to start working on his team at Infineon back in 2006. Ever since, he has spent countless hours supervising my work, helping me over obstacles, and proofreading. He has been a great mentor, team leader, and colleague. Thank you!

I would also like to thank my supervisors Professor Rückert, then of the University of Paderborn, and Professor Lavagno, of the Politecnico di Torino, for taking interest in my (at times, extensive) work. They always found the time to look after my progress and to give me academic counsel.

I extend my gratitude to my students Rafael Zuralski (Diplom), Jean Thomas (Diplom), Cunhua Xue (M.Sc.), and Dileesh Jostin (Intern). Especially without the help of Rafael Zuralski, who spent endless and late hours at the office working on the FPGA, the successful completion of the WLAN demo would not have been possible.

The collaborations with the University of Paderborn, TU München, and University of Dortmund (Oliver Hoffmann) were fruitful and important for my work. I am very grateful to Christian Liss along with the research group of Professor Rückert, who provided the FPGA prototyping system and invaluable and tireless support.

I especially acknowledge the firm support that I enjoyed at Infineon and later at Lantiq: Friedrich Geissler, my superior, who always had an open door for me, and Andreas Foglar, who ensured that my project could be completed. Charles Bry and Matthias Gries, who also helped proofreading this thesis. My colleagues including P. Kassel, G. Fenzl, A. Schmidt, and F. Beckmann, who were open and helpful towards me at any time. And Erwin Lock, to whom I am much obliged for his adept and patient technical help.

Last and not least, I would like to thank my dearest Rebekka for her endless patience and support, and to direct a "Danke" to my family and friends who have always believed in me and my work — in good and in bad times. You know who you are.

# Contents

# 1 Introduction

Ever since the broad adoption of the Internet, online and multimedia applications have evolved constantly and become ubiquitous in our homes. Applications range from web browsing to television (IP-TV), telephony (Voice-over-IP, VOIP), and online gaming. At the same time, the household itself has become connected. *Home networks* distribute data and services, e.g., from DSL lines to terminals such as phones and computers. Devices inside the house are connected to each other, such as media centers and gaming consoles. This trend is bound to increase, as devices controlling power consumption amount to a *smart grid*, or network convergence progresses.

This evolution towards comprehensive connectivity puts **increasing requirements** on home networks and on the devices (*Network Nodes*) it consists of. Key concerns include:

- *Throughput and efficiency* – The amount of data that can be transported over a (wireless) link is limited. Especially high-quality videos and large file transfers are demanding. Thus, efficient use of bandwidth is vital.

- *Quality-of-Service (QoS)* – Guarantees a level of service, i.e., throughput, latency, and reliability that is robust and sufficient for sensitive applications such as video and telephony. QoS requires precise scheduling inside the network and its nodes.

- *Security and usability* – In open networks, traffic must be encrypted and securely authenticated. Network setup must be easy in home networks, but advanced mechanisms are needed, e.g., for centralized management in offices.

- *Coexistence and reach* – Existing infrastructure must be reused but other networks must not be influenced. The range of networks is limited due to signal attenuation and other impairments such as unfavorable room geometries and interference.

These concerns have fostered an enormous development of *wireless* and *wireline* communication techniques. As extra wiring is often not an option, the physical *medium* must be either the air or reuse existing cables (electricity, television). In fact, such *shared media* often exhibit similar **adverse conditions** that conflict with above-mentioned requirements:

- *Shared and open* – Wireless transmissions cannot be confined to the intended area of use, or wired networks galvanically isolated from neighboring networks. Multiple networks and interfering devices thus occupy the same medium.

- *Dynamic and unreliable* – Random or periodical interference and distortions cause changing medium conditions. Thus, communications are highly unreliable.

- *Scarce and congested* – Unlicensed frequency bands as commonly used in home networks are scarce, e.g., in the 2.4 GHz range. Thus, multiple communication networks may compete for access to the same medium.

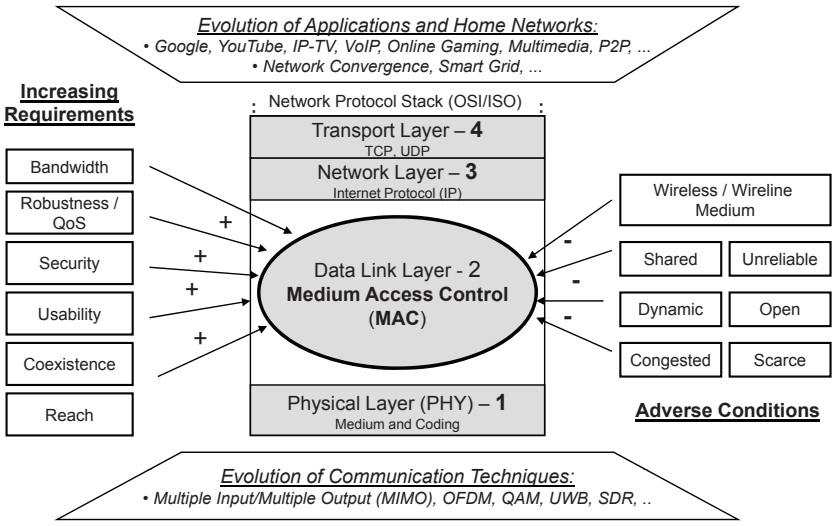
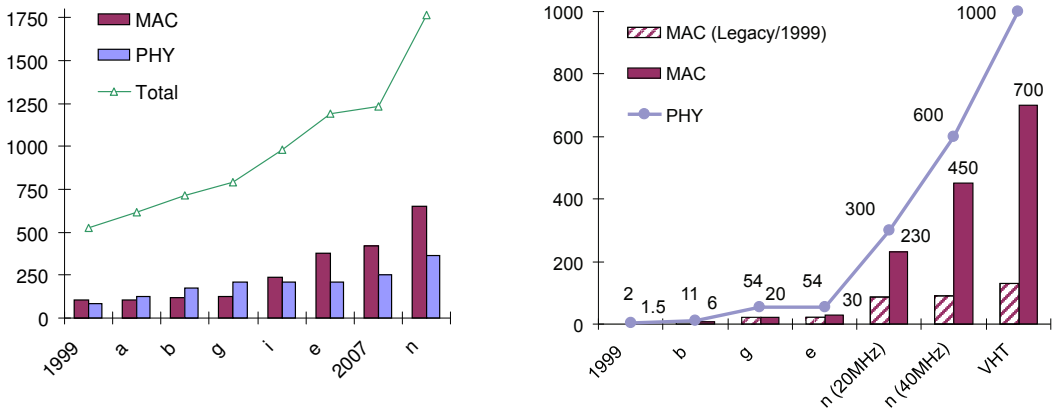



***Figure 1.1*** *– The Media Access Control (MAC) challenge: Increasing requirements collide with adverse effects posed by unreliable and limited communication media. This has increased the complexity of protocols at the central MAC layer significantly.*

The antagonism of increasing requirements and adverse conditions leads to the **Medium Access Control (MAC) challenge** of Figure 1.1. Networks are organized in *layers* according to the OSI protocol stack. The *physical (PHY)* layer is responsible for transmission, e.g., over a radio. It deploys advanced techniques such as *MIMO* and has evolved significantly to cater increasing throughput requirements. Now, the *MAC layer* plays a central role in the network stack: It must timely access, control, and configure the PHY in order to handle unreliable transmissions and to compensate for diverse transmission errors. At the same time, it must provide a reliable link service to the upper network layer, satisfying again increasing requirements in terms of throughput, QoS, and coexistence.

The consequence is an ongoing evolution of MAC protocols, as indicated by Figure 1.2 for the *IEEE 802.11 Wireless LAN (WLAN)* standard [86]. The number of pages specifying the MAC layer – and hinting at its **complexity** – has increased by 6X since the introduction in 1999, twice the increase of the scope of the standards document as a whole. In fact, standardization is often still in progress during development (**evolving protocols**). At the same time, PHY data rates have increased by 300X within 10 years. This does not only impact the MAC layer directly, but also underpins its importance: Without adequate improvements at the MAC layer, medium efficiency drops below 10 % to only 60 Mbit/s of effective throughput despite the medium's theoretical capacity of 600 Mbit/s.

Home networking devices often implement a whole family of such shared-medium protocols and are subject to **tight market competition**, requiring fast time-to-market and solutions that are cost-efficient in terms of chip area and development effort. The increase of complexity and evolving protocols at the MAC layer thus have a great impact on the internal device architecture and their development. *Medium Access Controllers (MACs)*, i.e. the subsystems implementing the MAC-layer protocol are deeply embedded into these systems. In the past, MACs have often been neglected due to their low complexity (e.g., for Ethernet) and realized in an ad-hoc fashion with inflexible architectures for chip area cost reasons. The radical increase in complexity, however, has put MACs into the critical path of the development




**(a) Number of pages** specifying the IEEE 802.11 standard [86] for different versions

**(b) Throughput (Mbit/s)** at the MAC and PHY layer with/without MAC optimizations

***Figure 1.2*** *– Evolution of the Wireless LAN standard: The complexity especially of the MAC layer has increased as indicated by its specification (a), also to improve channel efficiency in the face of higher capacities at the PHY layer (b).*

process and can lead to inefficient implementations. If protocol features are not completely understood or evolve during development, inflexible systems may be rendered useless, necessitating costly re-designs.

Thus, **three aspects** must be considered to develop new and competitive devices for evolving home-networking protocols in a productive way: (1) Enable a better understanding of protocol **applications** and requirements despite the complexity of their descriptions and system functions. (2) Address shortcomings of existing, inflexible **architectures** and exploit similarities of related protocols. (3) Improve on prevailing ad-hoc **development** methods. These aspects are further introduced in the following and will lead to a unified concept (Sec. 1.4), the objectives of this thesis (Sec. 1.5), and the thesis outline (Sec. 1.6).

## 1.1 Home Networking Protocols

The **MAC layer** in networking protocols provides a logical communication channel for the reliable exchange of *Protocol Data Units (PDUs)* or *frames*. It comprises addressing and channel access schemes that enable multiple network nodes to communicate within a multi-point network. As in other networking applications, the internal function comprises concurrent, dataflow-like processing of frames and their *protocol headers*. However, the system function – and with it the protocol description – has recently become much more complex, especially for shared media. Thus, such protocols are very difficult to understand and implement, and a new approach is needed to master this challenge.

Amongst the multitude of shared-medium home networking protocols, *IEEE 802.11 Wireless LAN* [86] has become a common and widely used standard. An emerging protocol is *G.hn*, which is claimed to support "every wire in the home" [81, 174] including, e.g., power lines. Other, short-range wireless communications are *Ultra-Wideband (UWB)* above 6 GHz (e.g., Wireless USB) or line-of-sight transmissions at 60 GHz. Strong similarities exist between

these protocols, as they all exhibit the properties outlined in the preceding section. Similarities have also been discussed in [128, 207]. In fact, the most recent *IEEE 802.11n* version of WLAN [89] is especially demanding in terms of complexity and dynamic behavior [234], and its throughput and timing requirements are similar to other protocols such as G.hn (see also Tab. 1.1 on page 8). Consequently, we regard **Wireless LAN** as a **demanding and representative shared-medium MAC protocol** and hence use it as our design driver throughout this thesis.

The WLAN protocol has **characteristics** that are typical for shared-medium protocols and that make them special. Due to the shared medium (the air), transmissions work in half-duplex mode only. The medium is highly unreliable due to reflections, dynamically changing environments (e.g., moving objects and stations), and interference with other wireless technologies. WLAN MACs consequently rely on precise timing in the microsecond range to detect and repair transmission errors and distinguish priorities among participants of the network. WLAN follows a channel access scheme called *Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA)*. It avoids collisions (e.g., by reservation timers) rather than recover from them, as collisions cannot be detected during transmissions (unlike Ethernet). Since the medium is unreliable and its quality is subject to variations, WLAN stacks spend considerably more time with exchanging station state than protocols for wired networks. This has led to the definition of diverse frame formats and functions for data transfers, control, and management of the network. Efforts to increase channel utilization have furthermore added sophisticated functions, e.g., for frame aggregation.

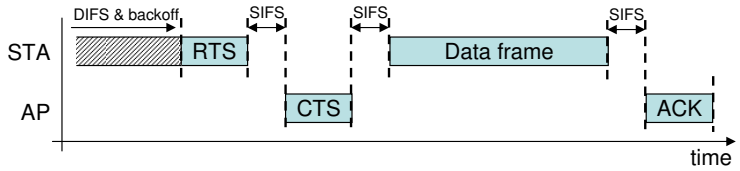



***Figure 1.3*** *– Frame exchange sequence in IEEE 802.11 Wireless LAN, imposing tight real-time response time requirements, e.g., of SIFS = 16 µs on the networking device.*

A basic frame exchange is shown in Figure 1.3. Assuming the network nodes, i.e., the *Access Point (AP)* and a client *Station (STA)* are synchronized, a channel reservation (*Request-to-Send, RTS*) is transmitted after a randomized back-off period. If no collision occurred, the AP signals *Clear-to-Send (CTS)* after a defined *Short Inter-Frame Space (SIFS)* of only 16 µs. Once the channel is reserved, the data frame is transmitted. After the AP has received and verified the frame, an *Acknowledgment (ACK)* is sent back to the sender. The WLAN protocol is further introduced in Chapter 2.

Given the complexity of the WLAN protocol and its sophisticated functionality under real-time requirements, a major problem arises from the standard document itself: it is in textual form, neither executable nor unambiguous, and covers a wide range of configuration options. It thus is tedious to understand, and capturing protocol knowledge is nearly impossible for non-experts (cf. Fig. 1.2a). The impact of features on required performance, implementation complexity, and architecture is not clear. Since the computational complexity of functions is not known, real-time constraints are difficult to verify. This makes it difficult to develop and explore architectures and to assess their dimensioning. Hence, a systematic approach is needed that puts the protocol (application) into focus and relies on an **executable benchmark** and associated reference configurations.

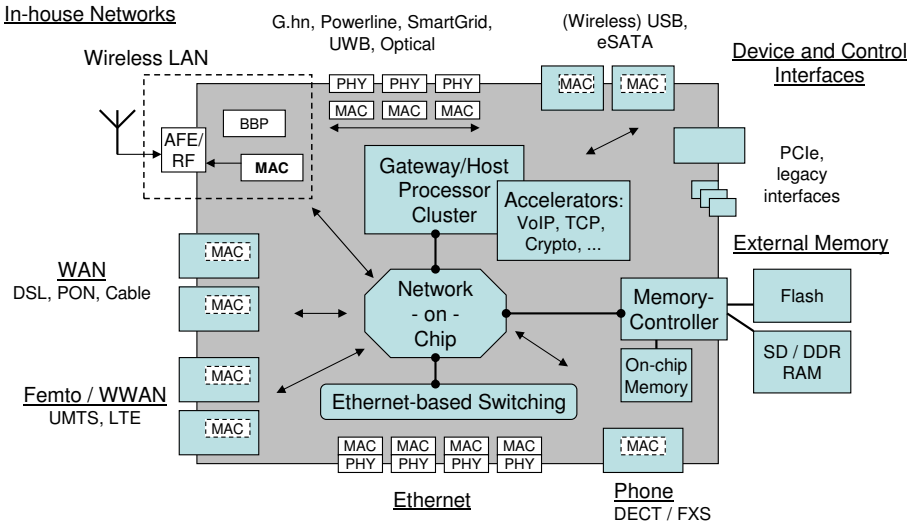



***Figure 1.4*** *– A residential gateway connects in-house networks with the Internet and provides, e.g., telephony services. The (exemplary) architecture shown comprises processors, interconnect, and a host of IO interfaces, most of which include MAC functions.*

In addition to ongoing standard evolution during development, recent protocol versions have introduced features that are mutually exclusive or overlapping, and their precise interworking may only become apparent during development or in the field. This necessitates late changes and updates. The consequence is that **flexible** device architectures are required. In fact, the diversity of functions may even require a programmable solution.

Worsening spectral congestion, e.g., in the 2.4 GHz bands, will soon necessitate increasingly intelligent and cognitive devices [97]. These devices may dynamically select the best available network and protocol to utilize even small gaps in the spectrum. At the PHY layer, this has led to *Software-Defined Radios (SDRs)* [163, 192]. SDRs perform all signal processing in software (SW), only relying on a minimal set of hardware (HW) components for analog/digital conversion and radio transmission. They can be programmed to support multiple communication standards in the same device. Thus, these efforts will have to be complemented by highly flexible MAC architectures [172].

## 1.2 Today's Home Networking Devices

A multitude of communication standards and thus a **heterogeneous landscape of MAC protocols** is found in *Customer Premises Equipment (CPE)*, i.e., in the devices of the home network. The central device is the *Residential Gateway (RG)*. It connects to the Internet through a *Wide-Area Network (WAN)* such as DSL or wirelessly (e.g., LTE). Inside the house, the RG distributes services over short-range *Local Area Networks (LANs)*. Especially WLAN has replaced cable-based Ethernet and become the de-facto standard. Emerging protocol include G.hn, *Powerline Communications (PLC)*, and UWB. DECT is increasingly supported for in-house telephony. In addition, RGs are becoming increasingly multi-functional, e.g., offering network storage, telephony, and firewalling.

This heterogeneity is reflected in the **internal architecture** of today's RGs, as exemplarily shown in Figure 1.4. RGs are complex embedded systems, that are increasingly integrated to a single *Systems-on-Chip (SoC)*. They comprises a large number of distinct, dedicated network interfaces. All building blocks are connected via on-chip interconnect such as a *Network-on-Chip (NoC)*. A processor cluster performs management services and *host functions* such as running interface drivers. It is aided by accelerators, e.g., for TCP offloading. The components of an interface are shown in greater detail for WLAN. They comprise the *Medium Access Controller (MAC)*, which connects to the rest of the system, and the *Baseband Processor (BBP)* implementing the digital part of the PHY layer. The analog *Radio Frequency (RF)* parts may be off-chip.

All communication interfaces comprise **MACs or MAC-like functions** that can be of considerable complexity and partly follow evolving protocols leveraging shared media. MACs therefore account for a significant part of the system, which both imposes special **requirements** on them but also holds great **potential**: CPE devices (and thus their embedded MACs) are subject to tight commercial constraints that demand for fast time-to-market, low development cost, and small chip area. Their internal architectures must provide the flexibility needed for complex or evolving protocols. At the same time, similarities exist in between MAC protocols, e.g., in terms of medium access or encryption.

**Today's existing MACs**, however, fail to address these market requirements, lack flexibility, and do not lend themselves to the exploitation of similarities. Instead, the MACs of an RG are a heterogeneous collection of closed, inflexible black-boxes that are buried deeply in their respective interface implementations. Their architecture is specialized and most of their function is hardwired. Their development follows low-level and ad hoc methods. This has the following **negative consequences**:

- *Redundant, monolithic systems* – Synergies cannot be exploited and resources are replicated redundantly, as the MAC subsystems are closed and their architecture is not modular. Lack of suitable representations of both the MACs and the overall system hinders architectural exploration, thus leading to non-optimal systems. Subsystem integration is inefficient and bottlenecks arise, as function splits and, e.g., routing between interfaces cannot be adjusted to system-level needs.
- *Systems that are difficult to develop and maintain* – Each MAC can only be programmed – at best – in a specific, low-level language, and MACs strongly rely on dedicated HW. The consequence is a multitude of different subsystems and associated tools that are inflexible and hard to program. Thus, development itself is tedious and time consuming, and the efforts for maintenance and adjustments to customer needs increase. Depending on the number of units, such Non-Recurring Engineering (NRE) costs may outweigh the chip-area cost advantage of more specialized systems. For evolving protocols, inflexible devices can even make complete redesigns necessary.
- *Dedicated systems* – Protocols may have configuration options that vary, e.g., in dimensioning and throughput. Or devices may target different markets. But hardwired, heterogeneous systems are difficult to scale, both in terms of performance and function, leading to expensive redesigns. For the development of related protocols, previously acquired knowledge and building blocks are lost or very difficult to reuse, leading to increased development times.

**Future MAC architectures** must therefore be open, modular, and amendable to exploration. Architectures should be centered around programmability and provide a common, easy-to-

use, and high-level programming model. This tackles the needs for adaptability and the realization of complex, diverse functions, but also leads to more homogeneous architectures that are easier to scale and reuse. As a consequence, careful hardware/software trade-offs are needed to devise efficient configurations.

Efforts are being undertaken to take advantage of and manage heterogeneity of MAC functions and physical media, but these tackle the problem above the MAC layer and still regard the MAC as a black-box. For example, the *Home Gigabit Access Project OMEGA* [96] relies on redundant network links to exploit the varying degrees of susceptibility to disturbances of different media. Thus, a convergence layer above the media-dependent MACs is introduced – the *InterMAC*. From an architectural point of view, however, it will only add to a system's heterogeneity, unless addressed in a uniform MAC approach.

## 1.3 Development and Exploration of Home Networking Platforms

Developing an embedded system such as a residential gateway or even a WLAN MAC subsystem is a complex undertaking under time-to-market and cost pressure. The increasing role of software has led to **Multiprocessor Systems-on-Chip (MP-SoCs)** for packet processing that combine multiple processors with application-specific accelerators, memories, and interfaces. Such MP-SoCs are very difficult to develop [148], to program [117], and a careful exploration of architectural trade-offs is vital for cost-efficient designs.

This is especially true for MAC development, where a seamless approach is missing and a multitude of pointed, low-abstraction tools are often combined and deployed in an ad-hoc fashion. And **IEEE 802.11n WLAN** is **especially challenging** in three regards:

- *Application Complexity Challenge* – Understanding the protocol standard and relevant features is difficult and calls for a careful domain analysis. Since the system function comprises concurrency, shared state, and precise timing, a capable and comprehensive modeling approach is needed. In addition, software optimization and debugging gain importance, necessitating means for performance feedback and analysis. Complex application state entails the need for long test runs.
- *Real-time Challenge* – Precise timing in the low microsecond range is at least comparable with related protocols and domains (see Tab. 1.1). Correct protocol behavior may depend on the timing behavior of the platform, which is related to, e.g., memory latencies. Thus, real-time constraints must be assessed comprehensively.
- *Productivity Challenge* – Market pressure necessitates fast development and good maintainability. This demands for productive design exploration and reuse on all levels. A comprehensive and seamless methodology for hard- and software is needed that offers appropriate abstractions and a path to implementation.

Recent efforts try to counter this development challenge ranging from design entry – *modeling* – to implementation – *deployment* – at the **Electronic System Level (ESL)** [204]. Early *design exploration* is considered a key aspect that promises fast turn-around of exploration steps, thus finding better architectures earlier. It necessitates to predict performance and dependencies from somewhat abstracted models in the face of dynamic workloads and complex interactions with shared resources such as memories.

***Table 1.1** – Real-time software challenges.*

| *Application / Domain* | *Response Time* |
|---|---|
| Production process in factory | 1,000,000 $\mu$s |
| Automotive applications (airbag, ABS, motor control) | 1,000 $\mu$s |
| Modern computerized tools (e.g., CNC) | 125 $\mu$s |
| **MAC layer:** G.hn home networking, powerline medium [174] | 20.5 $\mu$s |
| **MAC layer:** Energy-efficient Ethernet (EEE), low-power idle | 16.5 $\mu$s |
| Medical imaging applications (PET, MRI, Ultrasound)[147, 146] | 10 $\mu$s |
| **MAC layer: IEEE 802.11n Wireless LAN** | **2 – 16 $\mu$s** |
| Military applications (PATRIOT missile control) | 0.001 – 0.1 $\mu$s |

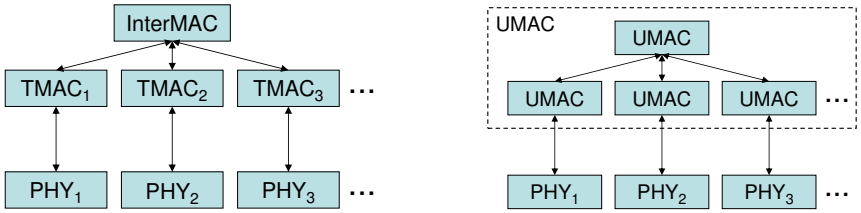
Exploring the MACs' architectural **design space**, trade-offs between flexibility, complexity, and cost arise. Complex, evolving protocols make programmable solutions indispensable. But this contradicts the need to cut chip area cost. Hard-wired systems are well suited for high-volume products and mature technology, or to address high-performance applications (e.g., [142]). For customer premises equipment, however, a good design must find viable trade-offs: identify performance bottlenecks that must be addressed by hardware (HW), but otherwise resort to software (SW) wherever possible. Such architectures comprise specialized blocks and application-specific and general-purpose processors, all relying on adequate interconnect and memory hierarchies, which unfolds several dimensions in the design space. Important design decisions concern:

- *HW/SW trade-offs* – What must be in HW for performance or cost reasons, what can be implemented in SW? What protocol-specific accelerators are needed? How many and what kind of cores are required? Can concurrency be exploited?
- *Memory Hierarchy* – How much code and data memory is required? How are packets buffered? Is fast on-chip memory needed? What is the performance impact?
- *Function split, dimensioning, and cost* – How is the MAC integrated? Where are expensive functions located? What is the chip area and the cost of a given feature?

## 1.4 The Promise of a Universal MAC Approach

We strongly believe that a new and more uniform approach to home networking protocols is overdue. This approach must address the concerns of the preceding sections: increasingly complex and evolving protocols, implemented on diverse, non-scalable, and closed architectures, and developed in an inefficient, ad-hoc fashion.

Hence, we propose a **Universal MAC (UMAC)** architecture and associated development approach [3], which puts the protocol application into focus. This means that a good understanding of the application domain is required, and that the application itself must be captured in an executable benchmark that unambiguously describes its function and reveals its performance requirements (e.g., realtime). Such a model can then be the basis for design decisions and guide implementation. A UMAC should thus be developed following an **application-driven methodology** such as [209] and complemented by a seamless flow that is optimized for shared-medium MAC protocols. And, during the development process, the following **architectural principles** must serve as guidelines towards a homogeneous and open MAC platform:




**(a)** Dedicated, closed (T)MACs. **(b)** Homogeneous UMAC.

***Figure 1.5*** *– The promise of a* Universal MAC (UMAC) *architecture.*

- *Protocol-agnostic* – avoid specialized functions and building blocks wherever possible and identify common MAC features.
- *Flexible* – be able to quickly adjust to new protocols and features during development and during operation on the field, relying on software where possible.
- *Easy-to-use* – support a unified and modular programming model that raises the abstraction level and opens the platform for changes. Provide a seamless framework for fast development and exploration. Prefer simple solutions.
- *Scalable* – be able to address differing requirements in terms of throughput, functions, and their complexity. A modular approach facilitates reuse.
- *Efficient* – be competitive (in terms of chip area or cost) with traditional architectures by allowing for HW/SW trade-offs based on systematic exploration.

The advantages of a universal architecture become apparent in Figure 1.5. Instead of a heterogeneous system *(a)*, a homogeneous node architecture *(b)* simplifies architecture development and eases deployment and reuse. A single UMAC instance (indicated by the dashed box) can even cater multiple PHYs at the same time. The potential of such an architecture – and also the productivity gain if complemented by a seamless approach – is further discussed as a conclusion to this thesis (Chap. 8).

## 1.5 Research Objectives

The objective of this work is to productively develop a universal MAC platform. Applying an established application-driven development methodology [209] to IEEE 802.11n WLAN as a demanding representative for shared-medium protocols, we propose:

- A ***seamless framework for development and exploration of MAC systems*** that overcomes limitations of existing frameworks. The most suitable framework and modeling language is identified and extended for productive use. The extended framework provides precise performance feedback and appropriate abstractions for the memory hierarchy, and is optimized for efficient early design exploration.
- A fully-functional ***IEEE 802.11n reference application*** and benchmark definition, which do not exist so far. Based on a careful domain analysis, this benchmark is used for quantification of dimensioning and features for a range of device configurations. It drives architectural exploration and opens a path to implementation.

- ***Exploration of an architectural MAC platform*** that is verified with a ***WLAN prototype***. The solution is based on a protocol-agnostic, fully-programmable multiprocessor platform and a modular streaming protocol for packet processing accelerators. MAC instances are customized based on careful evaluation of HW/SW trade-offs and optimization of the memory hierarchy. The prototype is amongst the first to comprise the IEEE 802.11n MAC layer and the only known realization of fully-flexible control frame processing in software.

## 1.6 Outline – Application-driven Development of a WLAN Platform

We will develop an IEEE 802.11n WLAN system starting from initial protocol analysis to prototypical deployment, following UMAC principles. This allows to assess the feasibility of a universal MAC platform and a systematic, productive development methodology. We deploy an **application-driven methodology** [209] that will enable us to devise UMAC instances for WLAN reference configurations and to develop a WLAN prototype:

1. *Application domain analysis* – identify and define representative and comparable system-level benchmarks and realistic workloads. The analysis includes the identification of common MAC functions, a discussion of current and future WLAN systems, and yields a set of reference WLAN device configurations (Chapter 2).
2. *Reference application development* – implement a performance-indicative yet architecture-independent reference application that captures essential system functions. A modular and executable IEEE 802.11n system model is specified in a domain-specific language that can be customized to the reference configurations (Chapter 3).
3. *Architecture-independent profiling* – derive architecture-independent properties by analyzing and profiling the reference application. The WLAN timing and the application's memory and performance requirements are precisely analyzed (Chapter 4).
4. *Platform exploration* – perform a search of the design space and define and customize the resulting platform architecture. The exploration follows the Y-chart [109] in that architecture and application are kept separated as HW/SW trade-offs, mapping, scheduling, and the memory hierarchy are optimized. Costs for reference configurations are derived and compared to existing WLAN systems (Chapter 6).
5. *Platform implementation and deployment.* An FPGA-based WLAN prototype implements an 802.11n access point, leveraging the exploration results and reusing the reference application as device firmware (Chapter 7).

For development and exploration we leverage *SystemClick* [9] – a simulation-based exploration and code generation framework. It is extended for productive use with MAC-layer protocols during all development phases (Chap. 5). Chapter 8 concludes on the usefulness and productivity of the chosen approach, the quality of the identified WLAN configurations, the main architectural features, and the resulting potential of a UMAC.

# 2 Wireless LAN Domain Analysis

As step one of the application-driven methodology towards a flexible MAC platform we perform an analysis of the application domain and its requirements. The analysis aims at exposing key architectural questions and to identify and define representative and comparable system-level benchmarks with realistic workloads. This will guide and reduce the complexity for subsequent steps, as the system's design space is narrowed down to a range of indicative device configurations.

We have selected *Wireless LAN (WLAN)* as a demanding domain representative in terms of complexity, throughput, performance requirements, and diversity of functions. The WLAN protocol deals with the unreliability of the wireless channel while aiming to provide efficient, reliable, and secure communications. This chapter gives a detailed overview of the functionality at the *Medium Access Control (MAC)* layer, focusing on aspects that are relevant for the architecture of a MAC implementation. The *Physical (PHY)* layer is discussed in terms of its impact on higher layers.

Precise timing sets the MAC layer apart from well-known throughput-oriented protocols such as Ethernet. A consequence are typical core MAC functions that are time-critical, and a number of extended MAC functions that can impact system cost and performance. As throughput increases in addition, this impacts system architectures and dimensioning of today's WLAN systems. We analyze approaches developed in academia and commercial WLAN devices and also evaluate trends for future wireless systems.

## 2.1 Wireless LAN Fundamentals

The *IEEE 802.11* standard [86] has been introduced in 1997, enabling wireless communication in local area networks (LANs). Initially the standard allowed for basic wireless communication in the 2.4 GHz band at bandwidths of up to 2 Mbit/s. **Standard additions** have then been made in order to increase throughput, security, and usability as is summarized in the *IEEE 802.11-2007* version of the standard [87]: The latest, *IEEE 802.11n*, was finalized in 2009 and offers (raw) data rates of up to 600 Mbit/s. These recent extensions are covered in the excellent book of Perahia [184]. In summary:

- *IEEE 802.11a* – 54 Mbit/s in 5 GHz bands (1999)
- *IEEE 802.11b/g* – 5.5 Mbit/s, 11 Mbit/s (1999) and 54 Mbit/s in 2.4 GHz bands (2003)
- *IEEE 802.11e* – Quality-of-Service (QoS) support and packet bursting (2005) [88]
- *IEEE 802.11i* – Enhanced security, WPA/WPA2 (2004)
- *IEEE 802.11n* – High Throughput (HT) using MIMO and aggregation [89]
- *IEEE 802.11o-z* – Planned amendments (discussed in Sec. 2.2.4 or, e.g., in [79]).

This section describes the MAC layer and parts of the PHY layer, focusing on aspects that are performance relevant and thus impact system architecture. The basic components of an *infrastructure* Wireless LAN include *Stations (STA)* that communicate over the wireless medium with a central *Access Point (AP)*, forming a *Basic Service Set (BSS)*. The access point, in turn, is connected to a *distribution system*, e.g., to an Ethernet or the Internet. Alternatively, stations can communicate directly with each other or form a meshed network. We recognize the **diversity of functionality** that must be supported in terms of data transmission, management, and medium control:

- *Data frames* can be aggregated, fragmented, transferred in bursts, encrypted, encapsulated with additional protocol headers, or stripped from protocol information.
- *Control frames* coordinate the medium reservation for atomic transfers and acknowledgments for data frames in a fine-grained manner.
- *Management frames* concern association with APs to gain network access, periodical monitoring of network properties, and extended reservation of the medium.

The coordination of **shared medium** access is based on randomized algorithms, where free air time is monitored and transmission opportunities are randomly selected. Priorities between traffic classes affect the waiting times for transmission after detecting a free medium, enabling QoS. Advanced packet aggregation mechanisms increase efficiency. All these features entail strict timing requirements in the low microsecond range.

### 2.1.1 Medium Access

Because of the shared nature of the wireless medium, access must be precisely timed and coordinated between stations[1]. If multiple stations transmit at the same time, a collision occurs. Importantly, typical radios are unable to detect collisions while transmitting. Thus, IEEE 802.11 uses a distributed *Carrier-Sense-Multiple-Access with Collision Avoidance (CSMA/CA)* algorithm, namely the *Distributed Coordination Function (DCF)*. Its operation is shown in Figure 2.1. Timing values for recent standard versions are 9 $\mu$s for slot time, 16 $\mu$s for *Short Interframe Space (SIFS)*, and 34 $\mu$s for *DCF Inter-Frame Space (DIFS)*.

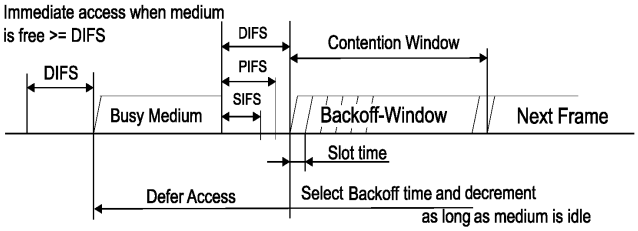



***Figure 2.1*** *– Basic medium access method using the contention window [87].*

DCF requires monitoring of the channel status for a DIFS interval prior to transmission. Typically, channel status is indicated by a *Clear-Channel-Assessment (CCA)*. If the channel is found busy, transmission is deferred. If multiple stations contend for the channel at the same time, they will simultaneously find that the channel is released and try to seize it. Collisions

[1]Unless special properties of the access point are relevant, the terms *station* or *client* is used interchangeably for any member of the wireless network including the AP.

are the consequence. DCF uses random backoff to avoid such collisions, forcing stations to defer channel access throughout an extra period: the contention window. The precise backoff time for each station is determined by:

$$\text{Backofftime} = \text{random}() \times \text{aSlotTime} \leq \text{contentionWindowSize}$$

Now collisions are restricted to stations picking the same transmission slot. In this case, the contention window is increased in steps to render repeated colliding accesses less likely. More advanced access mechanisms include Enhanced Distributed Channel Access (EDCA), see Section 2.1.3, HCF controlled channel access (HCCA), and the contention-free access method Point Coordination Function (PCF) relying on a centralized controller.

### 2.1.2 Basic Protocol Operation

Due to the unreliability of the medium, DCF includes a positive acknowledgment scheme. If a frame is successfully received by the destination, an *Acknowledgment (ACK)* control frame is sent to notify the source about the successful reception. Fragmentation of frames reduces the effect of bit errors due to channel distortions.

Additional protection is provided by a virtual carrier sense mechanism based on the *Network Allocation Vector (NAV)*. This allows to (virtually) reserve the medium for sequences of packets ahead of the actual transmission. For this, short *Request-to-send (RTS)* and *Clear-to-send (CTS)* control frames are exchanged between source and destination stations during the contention period, also mitigating the effect of collisions by reducing wasted channel time. RTS/CTS furthermore addresses the *Hidden-Node* Problem, where two stations are within reach of an AP but out of range with respect to each other.

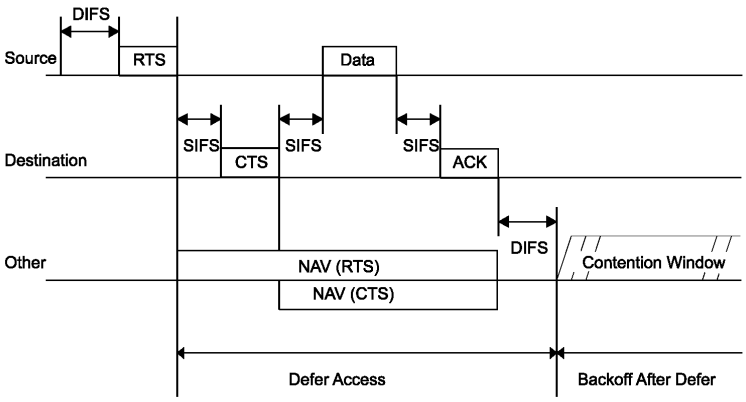



***Figure 2.2*** *– Frame exchange sequence in IEEE 802.11 Wireless LAN. The Network Allocation Vector (NAV) is also shown [87].*

Figure 2.2 shows an atomic frame transfer with medium reservation. The STA has gained access and sends an RTS to the AP. The AP replies after a defined interval (SIFS period) with CTS. The STA then starts the data transfer after another SIFS period. At the end, the AP confirms successful reception with an ACK frame, again after SIFS. This sequence is atomic. The SIFS periods guarantee uninterrupted operation since randomized channel access by other stations can only start after the longer DIFS period and a random backoff.

### 2.1.3 Quality-of-Service

The 802.11e standard extensions [88] introduced QoS support, leading to increased protocol complexity. The basic idea is to support four priorities called *Access Categories (AC)* with associated queues and per-queue EDCA functions. Collisions can now also occur internally in between ACs and will trigger the back-off procedure as well. A fixed offset delay period prior to the contention window, called Arbitrary Inter-Frame Space (AIFS), is added for lower priorities. This effectively guarantees better access times for high-priority traffic. In addition, the channel may be reserved by stations for extended periods by setting the NAV counter accordingly. These *Transmission Opportunities (TXOP)* reduce channel access overhead as no subsequent contention procedure is necessary. Furthermore, resource utilization for high-speed stations is improved, because these stations can now fit multiple frames into one TXOP. If a TXOP is assigned but no packets are left for transmission, a CF-END frame may indicate an immediate end of the TXOP (see Fig. 2.3).

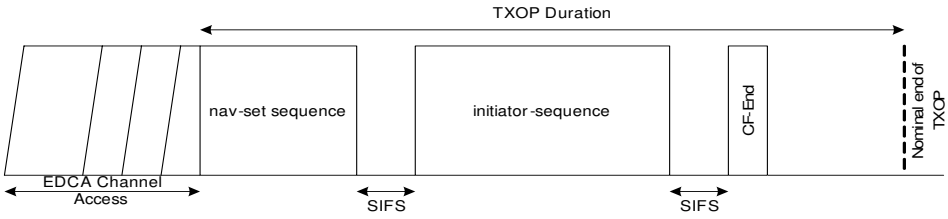



***Figure 2.3*** *– A Transmission Opportunity (TXOP) in IEEE 802.11e/n [87].*

### 2.1.4 High Throughput and Aggregation

The main objective of 802.11n is to increase effectively usable throughput. Increasing PHY raw data rates alone, however, is not sufficient: Channel efficiency is below 20 % for data rates above 500 Mbit/s due to significant overhead from frame encapsulation, inter-frame spaces, contention windows, and control frames. This is addressed by:

- *Block Acknowledgments (BlockACKs)* and *TXOPs*, reducing access/control overhead.
- *Packet aggregation*, reducing access and encapsulation overhead at MAC and PHY.

Control frames occupy a significant share of medium time because inter-frame spaces need to be respected in addition to the actual transmission. Furthermore, control frames are typically sent at lower data rates for stability and compatibility reasons. Thus, *BlockACK* frames are introduced that acknowledge multiple data frames with a single control frame (Fig. 2.4). They can either be requested explicitly by a *BlockACK Request (BAR)*, or implicitly along with aggregated frames. Data frames within TXOPs are sent back-to-back with only a SIFS time in between. These measures increase throughput but also complexity: Both sender and receiver must keep track of frames for re-transmission or for re-ordering. Also, TXOPs need to be planned and packets need to be scheduled accordingly.

Small packets incur a much higher encapsulation overhead than larger packets. At the MAC layer, every packet has a WLAN header and a CRC checksum and entails inter-frame gaps and acknowledgments. At the PHY layer, significant per-packet overhead is caused by preamble and PLC headers (cf. Sec. 2.1.5). This overhead is 8 % at 54 Mbit/s but increases to

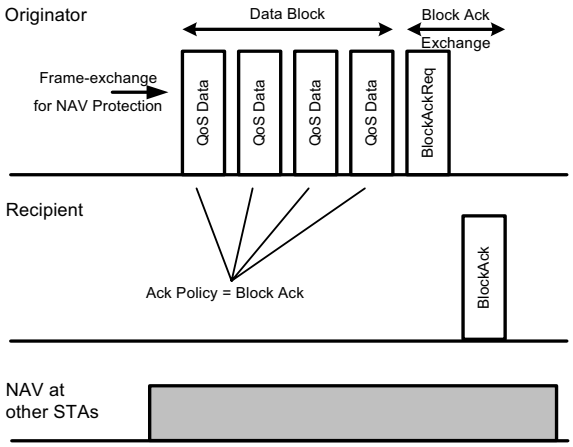



***Figure 2.4*** *– Multiple frames are acknowledged by a single immediate* BlockACK *[89].*

73 % at 540 Mbit/s for 1500 Byte packets [184]. Thus, small frames are aggregated into larger ones. 802.11n allows aggregation of up to 64 kB and 64 packets. For the sake of robustness and efficiency, however, packets are aggregated in two stages:

- At the MAC layer into *Aggregate MAC Service Data Units (A-MSDUs)*.
- At the PHY layer into *Aggregate MAC Protocol Data Unit (A-MPDUs)*.

A-MSDUs are opaque to subsequent functions and share encapsulation such as CRC. Thus, bit errors within A-MSDUs cannot be recovered. *Subframes* do not require a separate header. Instead a 16 B subframe header is provided for de-aggregation and to recover destination addresses at the receiver. As most MAC processing occurs per frame, A-MSDUs effectively reduce the processing load. This comes at the expense of extra functions and the need for an efficient aggregation strategy.

A-MPDUs also consist of subframes, but these are concatenated at the MAC/PHY interface. A subframe header stores the length for de-aggregation and is protected by CRC (see Fig. 2.5). The delimiter allows searching for subsequent subframes when a header was corrupted. A-MPDU subframes are independent of each other. Consequently, damaged frames do not affect other frames in the aggregate and the BlockACK mechanism must be used to keep track of failed transmissions. For the MAC, PHY-layer aggregation is a major source of complexity, as aggregates need to be planned and scheduled ahead of time in addition to reordering and replaying of corrupted frames. Thus, typical systems only support a limited number of A-MPDU streams.

### 2.1.5 Physical Layer

The physical (PHY) layer comprises the digital *Baseband Processor (BBP)* and the mixed-signal *AFE/RF radio front end*. It is responsible for frequency coding and transmitting frames over the air (see also Fig. 1.2b on page 3). This increase is due to advanced coding schemes (especially Orthogonal Frequency-Division Multiplexing, OFDM) and the use of Multiple-Input-Multiple-Output (MIMO) in 20 MHz and 40 MHz channels. Frame encapsulation in 802.11n PHYs depends on the *High-Throughput (HT)* operation mode, i.e., legacy (non-HT),

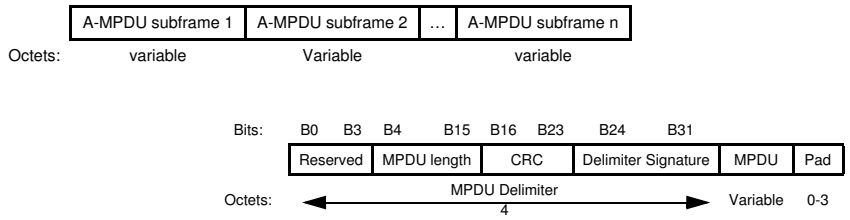



***Figure 2.5*** *– A-MPDU aggregation in IEEE 802.11n [89].*

HT-mixed, and HT-greenfield as shown in Figure 2.6. The preamble (STF/LTF in the figure) is fixed and needed for synchronization. The signal field contains basic frame information such as rate, length, and encoding. Depending on the number of spatial streams, additional fields are needed. Data are transmitted in symbols of 3.2 $\mu$s each that are encoded with one of 64 different *Modulation and Coding Schemes (MCS)*.

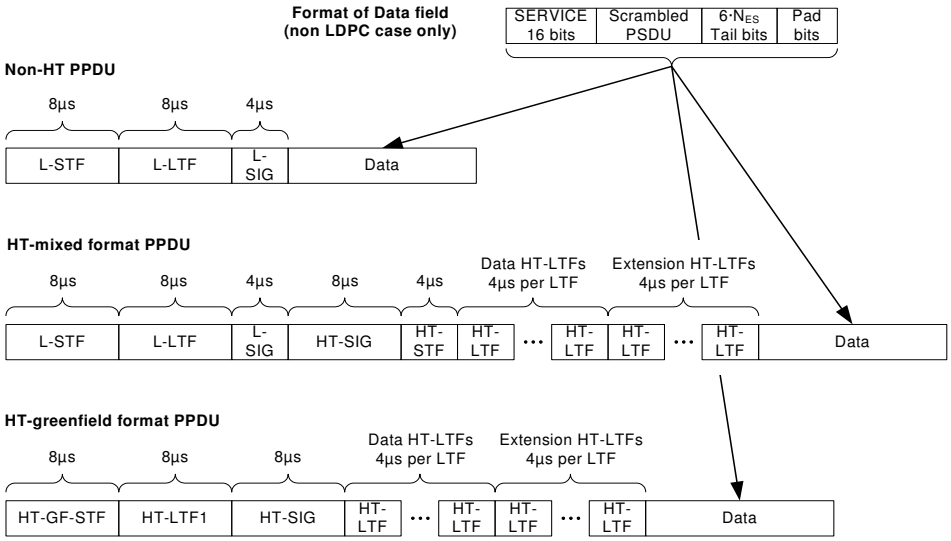



***Figure 2.6*** *– PHY encapsulation for legacy, mixed, and greenfield formats [89].*

The MAC-layer impact concerns timing (see Sec. 4.1), in-frame protocol fields, and fine-grained PHY control. 802.11n introduces a *HT-Control* field into the WLAN header. It is used for antenna selection, calibration, and to indicate sounding packets [184]. Spatial diversity (MIMO) must be controlled by the MAC and includes proper selection of coding and configuration for transmit beamforming. The performance impact of MIMO on the MAC is limited. However, fast link adaptation and rate selection for throughput maximization under QoS constraints have become a complex topic of research [38].

## 2.2 Current and Future WLAN Systems

Following the goal towards a flexible and scalable MAC platform, we first analyze common MAC operations in order to motivate function splits and architectural options. This analysis is kept as protocol agnostic as possible, such that it also applies to a generic UMAC architecture. Then, related work from academia and state-of-the-art commercial systems is evaluated. Trends and features are assessed that may influence future systems.

### 2.2.1 Core and Extended MAC Functions

Shared-medium MACs can cover a broad range of functionality, such as for WLAN in home networks, or be of small scale and energy optimized, e.g., in Wireless Sensor Networks (WSNs). Almost all such MACs have two things in common: Their operation is time-critical, and they rely on a basic set of functions. These **core functions** are typically deeply embedded and realized close to the PHY due to large communication delays with general-purpose or host systems. We generalize the views of Nychis [172], who identified core functions for SDR, and of Sauer [207]:

- *Precise scheduling* in the $\mu$s range is required for both time division (TDMA) and contention-based medium access and for maximum performance.
- *Carrier sense* is used by contention-based protocols for synchronization. The *ACK/nACK protocol* establishes a reliable communication link. The receiver acknowledges error-free packets by using a sequence number or triggers retransmission.
- *Back-off* is related to precise scheduling in the context of randomized rescheduling of transmission in a distributed scheme.
- *Fast Packet Recognition and Classification* is required to quickly detect incoming packets. Classification according to packet types may be based on multiple bit fields (e.g., address, format, type) of the header.
- *Frame integrity* must be validated, e.g., with checksums or error-correcting codes.
- *Dependent packets*, i.e., the fast generation of response frames is critical especially in high-performance, acknowledgment-based protocols or for channel reservation.
- *Fine-grained radio control* and access to physical layer information traditionally refers to power and rate selection, but is becoming more critical and complex with advanced MIMO implementations.
- *Serializing, framing, scrambling* is done by the PHY but may be aided by the MAC.

Specifically in the context of home-networking protocols, additional requirements such as high throughput, security, and QoS arise and entail more complex functions. These **extended functions** can be partly realized outside the MAC on a host or general-purpose system. But since these functions impact the performance and cost of the composite system, their consideration must be included in our analysis:

- *Packet buffering and interfacing* with multiple queues for QoS, aggregation, and power-saving schedules. Depending on the function split, queues and packet storage may be partly realized on the host and in external memory for cost reasons.
- Dynamic *memory management* for packets and their payloads, e.g., in segments.

- *Frame encapsulation* and addressing that, e.g., necessitate header translation.
- *Network integrity* based on cryptography (AES, WPA). In addition, WLAN includes measures to update crypto keys and to detect network attacks.
- *QoS classification and scheduling* to determine, e.g., user priorities and access categories and to prioritize traffic on the medium.
- *Aggregation and flow-control*, requiring advanced packet buffers that support selective retransmission, reordering, and keeping track of packets. Flow control, e.g., based on BlockACKs is demanding if directly dependent on received packets.
- *Dynamic planning.* Even without a central coordinator, network performance can be increased if channel reservation is properly planned, e.g., by adjusting TXOP duration and A-MPDU lengths dynamically in WLAN.
- *Configuration and management.* Concerns the identity of the device as determined during initialization, the negotiated link parameters, and user interaction. Access to internal state and reconfiguration is needed.

### 2.2.2 Device Types, Architectures, and Dimensioning

Depending on the device type (AP, station *Network Interface Card (NIC)*) and intended deployment (e.g., low-cost, integrated vs. standalone) different **function splits** between the MAC and the host system are possible. Some extended functions may be time critical and thus must be integrated to the core MAC. Examples include real-time flow-control, fine-grained RF control, and QoS. Also, extended functions can impose a high computational load on the host. We identify three MAC categories:

- *Lite-MACs* – are systems that implement only a minimum set of time-critical functions in the MAC and rely on a host for running extended functions in a SW driver. Open-source drivers exist for popular chipsets [93, 139]. Lite-MACs are typically found in NICs that handle only a single connection, or in very cost-optimized APs.
- *Full-MACs* – In integrated systems, the total performance and cost are important. Full-MACs thus implement all performance-impacting functions in the MAC. They may still rely on a host processor, e.g., for high-level yet low-performance management functions and to buffer packet data externally.
- *Standalone MACs* – Similar to Full-MACs, a Standalone MAC performs all functions including packet transfers and high-level management. Additional memory is required to buffer packets and to store application code.

***Table 2.1*** *– Function splits and device types.*

| | Lite-MAC | Full-MAC | Standalone |
|---|---|---|---|
| Core Functions | yes | yes | yes |
| Extended Functions | QoS/(crypto) | performance-relevant only | yes |
| High-level Management | no | no | yes |
| Dedicated Memory-IF | no* | optional | yes |
| Load on Host | yes | minimal | n.a. |

*) needed only if the host interface is very slow or has a large latency

Current WLAN devices differ in their **PHY configuration**, impacting system dimensioning. With the number of supported spatial RX/TX streams (MIMO) the throughput is increased as summarized in Table 2.2. MIMO also allows to operate multiple independent MACs in the same device, e.g., to keep legacy devices from slowing down the network. Having multiple independent MACs improves the effective throughput, as the relative per-MAC overhead is lower with reduced channel capacity. Not considered here are other combinations of RX and TX streams that are used to increase reliability and reach.

***Table 2.2** – Configurations of the PHY layer in IEEE 802.11n (40 MHz channel).*

| *PHY Config.* | *Eff. Utilization* / Capacity* | *Purpose / Deployment* |
|---|---|---|
| 1x1 | 120 / 150 Mbit/s | Low-end devices, integrates with other 802.11n equipment |
| 2x2 | 230 / 300 Mbit/s | State-of-the-art in 802.11n equipment |
| 3x3 | 340 / 450 Mbit/s | Next generation Access Points |
| 4x4 | 420 / 600 Mbit/s | High-end enterprise-level equipment |
| 3x3 + 1x1 | < 450 / 600 Mbit/s | Multi-MAC, independent operation, e.g., 802.11n + legacy |
| 2x2 + 2x2 | 460 / 600 Mbit/s | Multi-MAC, independent Access Points |
| 4x 1x1 | 480 / 600 Mbit/s | Time and space multiplexing maximizes efficiency |

*) effective utilization of the channel with full 802.11n optimizations, according to [184]

### 2.2.3 State of the Art

A **flexible** and **scalable** architectural MAC platform is required for evolving standards in home networking technology such as WLAN. Keeping the system **protocol-agnostic** and **homogeneous** allows reuse for related protocols. Thus, we favor a fully programmable, platform-based approach that systematically maps functions to dedicated hardware (HW) only if necessary for performance and cost reasons. Keeping complex operations such as control frame processing and channel access in software (SW) facilitates development and shortens time to market. **Ease of use** is increased by a high-level programming model and a seamless development flow. A systematic exploration of the full MAC function in indicative networking scenarios (see Sec. 3.4) is essential for **efficient** solutions and to estimate device costs. Functions must be split flexibly, depending on **commercial** and practical reasons. Finally, flexible MACs are interesting for SDRs.

From this perspective, we review the state of the art in flexible WLAN MAC architectures and systems from both industry and academia. Traditionally focusing on the PHY layer, the interest in fully-programmable MACs has been fostered only recently in SDR contexts and by evolving and more complex protocols.

#### Academic Approaches

To the best of our knowledge, no IEEE 802.11n MAC-layer architecture description is publicly available. Only few high-level models exist (see Sec. 3.1.3) that are restricted to facets such as frame aggregation and lack a path to implementation. Thus, the search is restricted to flexibility found in legacy or generic WLAN MACs and the applicability to related protocols. Systematic exploration and programming models are highlighted.

**Flexibility at/above the MAC Layer** Motivations in academia for seeking flexibility include protocol experimentation and cross-layer optimization. More recently, the use of heterogeneous network resources and scalability have also encouraged flexibility. The following approaches view the core MAC as a blackbox and would thus benefit themselves from an open and homogeneous architecture that can be programmed in a uniform way.

For example, flexibility is added to commercial Lite-MACs by bypassing inaccessible core MAC functions in the driver for practical reasons. The MadWifi driver is a popular choice due to its good availability. SoftMAC [169] supports different frame formats and control over transmission timing. MadMAC and FreeMAC [219] experiment with channel access and extensions to 802.11. FlexMAC [144] bypasses firmware functions to maximize software control. MadWifi can also be programmed using Click [131]. All these approaches fail at real-time response generation, and channel access can only be changed indirectly. Among the few who actually change the MAC is Grunenberger [72], who has access to the internal firmware and implements a modified DCF on off-the-shelf Intel IPW2915 NICs.

Efforts to add flexibility by establishing a layer 2.5 above the MAC layer are gaining attention in home networks. They aim to cope with network heterogeneity and to establish QoS over diverse links. Efforts include [247] for WLAN, and Multi-MAC [51], a cognitive radio based on SoftMAC implementing a set of protocols. The OMEGA project introduces an InterMAC layer [156] to handle a range of physical devices. Hoffman [80] maps multiple wireless protocols to a large NoC for scalability and QoS reasons.

**Traditional MAC Architectures** The core MAC architecture is traditionally hardware (HW) dominated for performance, cost, or power reasons. A common function split partitions the MAC into a *Low-MAC* HW accelerator, and a *High-MAC* running as software (SW) on a processor. The Low-MAC comprises processing-intensive data-flow operations and also real-time response generation, channel access, packet buffering, and aggregation functions. This results in limited flexibility.

Many authors (and most commercial WLAN systems) follow this **usual HW/SW split** but do not quantitatively justify their design decision. These include:

- Bernasconi [24] presents a straightforward implementation of 802.11 on a DSP and maps time-critical functions to FPGA hardware. Similarly, Fujisawa [62] designs an ASIC following the usual split and uses an 80 MHz CPU for the High-MAC.
- Hannikainen [75] starts with a SW description of the MAC layer in SDL. However, the use of the SDL runtime makes it impossible to realize Low-MAC in SW and necessitates a dedicated HW accelerator.
- Even commercial prototyping platforms for WLAN such as Signalion's SORBAS encourage the use of dedicated HW for some core functions [225]. Still, the architecture proved suitable to evaluate MAC protocol extensions [232].

Few publications actually **analyze performance requirements** of real-time functions during architecture design **quantitatively** and implement these in dedicated HW, including:

- Panic [179] starts with an SDL simulation model and profile generated C-code together with the SDL runtime. Even with optimizations he observes excessively high performance requirements for 802.11a and therefore discards a fully-flexible implementation. Further quantitative insights that would support the function split of the resulting system are not provided.

- Samadi [203] states that 4 $\mu$s are available for MAC response generation (unlike our findings of Sec. 4.1.1), and control frames and channel access must thus be in HW. RTOSes are evaluated for the High-MAC but discarded due to their large overhead.
- Masselos [152] systematically explores the MAC layer for the related HYPERLAN/2 protocol, but concludes that real-time processing must be mapped to dedicated HW.
- Power and performance constraints for 802.15.3 MACs motivate Dietterle [49] to follow the usual split. Expensive and time-critical tasks are handled by an accelerator integrated on-chip with a 32-bit processor to reduce the operating clock frequency.

Others focus on HW implementations of low protocol functions only. An efficient implementation of 802.11b is presented in [248]. Haroud [77] uses SDL for development and verification of an accelerator for UWB. Complete packet processing pipelines can be synthesized [224], but this approach ignores the complexity of advanced MACs. Finally, dedicated HW can be used to achieve specialized functions, e.g., in real-time Ethernet [142].

**Flexible MAC architectures** Recently, flexible MACs have attracted more interest [172, 78, 229]. In addition, many prototyping and SDR platforms exist [108, 98, 78]. Efforts to implement basic real-time functions of the WLAN MAC layer flexibly have often failed in the past due to inefficient runtime environments, e.g., for SDL [179, 75], the use of RTOSes and interrupts, or communication latencies:

- Shono [220] implements a basic 802.11a MAC fully in SW on a PowerPC at 400 MHz. It runs a VxWorks RTOS and is part of an SDR prototype with several physical modules connected by a common bus. Features such as security and RTS/CTS frames are lacking, and timing constraints are missed by one order of magnitude.
- A protocol similar to 802.11 is modeled in UML, and mappings to a scalable MP-SoC platform are explored in [15]. Response frame generation in SW, however, fails to meet low $\mu$s real-time budgets due to RTOS overhead.
- Based on the USRP platform for SDR, Dhar [47] implements response generation in domain-specific language Click on the host system, but fails timing requirements due to communication overhead.

More recently, truly flexible **SW-based architectures** have emerged. However, they handle only basic protocol operation and partly rely on high-performance platforms.

- Lee [126] uses an application-specific SoC to implement a subset of 802.11b. The architecture is heterogeneous: A general-purpose core with caches is aided by a *supplemental* processor generating real-time responses. The MAC cycle budget analysis of 5 $\mu$s is pessimistic compared to our analysis (Sec. 4.1.1).
- Nychis [172] states response generation as a general problem in MAC architectures. He addresses the problem of latency on the USRP platform by implementing core functions close to the PHY in the GNU radio environment. GNU radio, however, suffers from drawbacks regarding MAC development [47].
- Hedde [78] uses an FPGA to implement High-MAC functions based on the MadWifi driver on a PowerPC running Linux. A dedicated Microblaze core performs real-time operations. Requirements are met for basic response handling with a relaxed slot time of 20 $\mu$s but, e.g., encryption is missing. The need for an optimized MAC/PHY interface is highlighted that yields 10 $\mu$s for MAC processing.

- Tan [229] meets real-time requirements for basic 802.11 frames in the SORA SDR approach. He takes advantage of the high performance offered by a commodity multi-core PC, dedicates a core to real-time processing, and optimizes communication latencies with the RF hardware. In addition, prediction and pre-calculation are used for timely generation of frames.

Flexibility can also be based on **reconfigurable logic.** Nabi [166] chooses the usual high-/low function split, but proposes reconfiguration to adapt to different protocols on the fly. Not especially tailored to WLAN MACs are efforts to map (Click) models directly onto FPGAs [116, 201], or to build reconfigurable network processors [173] and NICs [215]. However, FPGAs have not made their way into commercial consumer electronics for cost and power reasons and are thus no alternative to flexibility based on SW.

**Programming Models and Modeling Languages** Only few approaches to MAC development offer a high-level programming model or leverage a modeling language (see Sec. 3.1). However, these often fail to yield efficient solutions or target higher layers only:

- *SDL* – the widely distributed SDL model for 802.11a motivates, e.g., the work in [179, 75], but the results are inefficient due to the SDL runtime.
- *UML* – Arpinen [15] models a protocol similar to 802.11, but requires an RTOS and does not meet real-time targets.
- *GNU radio* – efforts based on the USRP platform [47, 172] yield efficient implementations, but GNU radio has some drawbacks [47].
- *Click* – Existing efforts include a Madwifi driver implementation [131] and combinations with GNU radio [47], but core and real-time MAC functions are not covered successfully so far.

### Commercial Systems

The 802.11n version of WLAN was finalized in 2009 and has now become the standard in consumer home-networking devices. Commercial vendors of chipsets include Atheros [185, 206, 155], Broadcom [35], Marvell [149], Intel[93], Lantiq[121], Ralink [191] and many smaller IP vendors such as Wipro [245], Metalink, and Redpine. Their focus is traditionally on the PHY, which occupies a larger chip area than the MAC, and on the overall system. We observe the following **trends and architectural decisions**:

- *MIMO* and advanced coding are increasingly important at the PHY layer, improving throughput and robustness [184]. Devices range from low-cost or mobile 1x1, state-of-the-art 2x2, and high-end 4x4 configurations (see Tab. 2.2). Dual-band (5 GHz) devices benefit from less interference.
- *Device integration* of digital with mixed-signal parts and external interfaces (e.g., USB) reduces cost, as feature size decreases [114]. Residential gateways often integrate WLAN system as a sub-module.
- While Lite-MACs were common in NICs and low-cost APs, the interest in *Full-MACs* is rising with more complex protocols in order to offload the host [191, 35]).

- Dedicated *memory interfaces* are becoming more common, as throughput increases, low access latencies are required for QoS reasons, and slow host interfaces such as USB need to be compensated for.

However, also distinctive features of the MAC itself are gaining attention as new and more complex protocol versions offer more possibilities. These **protocol features** have a strong impact on the evolution of MAC architectures:

- *QoS* support enabling low latency applications by implementing admission control such as Wi-Fi Multimedia (WMM). Advanced MAC schemes based on central coordinators (PCF) or TDMA are only found as proprietary solutions (e.g., [149].
- Fast *link adaption* exploits the plethora of channel configuration options (channel selection, coding, beam forming) to increase throughput and reliability. Support for *advanced features* such as RDG, power saving, and TXOP continuation (see Sec. 2.1).
- High *throughput* by exploiting, e.g., aggregation. However, throughput differs only marginally in current systems [237] under good channel conditions. Close to no differences exist for *security*, as WPA2 based on AES-CCMP has become standard.
- *Dimensioning* in addition to throughput includes the number of supported clients, BlockACK streams, and QoS categories. *Multi-MAC* operation (see Sec. 2.2.2) and *legacy* modes increase usability.

Above features including complex scheduling, dynamic adaption, and advanced functions require a programmable and real-time capable system. Reconfiguration for different protocols favors SW-based implementations, and together with dimensioning demand for a careful exploration. However, little is known about **architectures of commercial WLAN MACs**, their flexibility, and their development:

- Most available information, e.g., from product sheets [191] focuses on *device configuration and features*. Publications such as [185, 206] are limited to high-level descriptions of 802.11n MACs, but give quantitative chip area results (see Sec. 6.8.4).
- From analyzing feature lists and block diagrams, it can be concluded that many systems are still built on inflexible architectures following the *usual HW/SW split*, e.g., [149]. A-MPDU and BlockACK operation is typically in HW (e.g., [191, 121]).
- Some systems/architectures claim to be more *flexible and scalable*, such as Wipro [245] implementing an 802.11bg Full-MAC based on an ARM7 SoC.
- Few designs are known to use *SW for Low-MAC functions*. The Intel IPW2915 executes firmware on an embedded core for Low-MAC functions, but no architectural details are given [72]. Lantiq [121] uses a small core, e.g., for response generation and channel access, but still relies on HW support, e.g., for queuing and BlockACKs.

While architecture internals remain mostly unclear, we expect an increasing **trend towards flexibility**. The usual HW/SW split is mostly used for legacy protocols or low-power devices, but often is motivated by rule-of-thumb decisions and gradual system evolution. Many commercial WLAN systems even fail to meet proper timing [26]. However, the advantages of easy-to-use programmable platforms become apparent with increasing complexity and and network evolution. Programmable platforms will improve initial time to market and can be tailored to special use cases and related protocols. In addition, decreasing feature sizes and increasing chip integration diminish the cost impact of the digital MAC. Thus, costs for design and maintenance can easily outweigh chip costs.

***Table 2.3*** *– Overview of related work on Wireless LAN MACs.*

| *Author/Title* | Protocol /M,P | Prog. Model | Flex./SW MAC | Product/11n | SDR/UMAC | Scalability | Systematic DSE | Quant. Results |
|---|---|---|---|---|---|---|---|---|
| Neufeld[169]: SoftMAC | 11g MAC | – | high-l | no | no | no | no | no |
| Letor[131]: nsclick | 11g MAC | Click | high-l | no | no | no | no | no |
| Doerr[51]: Multi-MAC | 11g MAC (ext.) | – | high-l | no | yes | no | no | no |
| Meyer[156]: InterMAC | multi, MAC 2.5 | – | no | no | yes | no | no | no |
| Hoffman[80]: Scalable NoC | multi, M+P | – | no | no | part. | yes | part. | no |
| Fujisawa[62]: Single-Chip | 11a, P>M | – | usual | no | no | no | no | area |
| Hannikainen[75]: Using SDL | TUTWLAN | SDL | usual | no | no | no | no | part. |
| Dietterle[49]: HW-accel. | 802.15.3 MAC | – | usual | no | no | no | part. | part. |
| Panic[179]: SoC Impl. | 11a MAC | SDL | usual | no | no | no | no | part. |
| Samadi[203]: Novel Impl. | 802.11 MAC | SDL | usual | no | no | no | part. | no |
| Masselos[152]: Wireless | HyperL. MAC | – | usual | no | no | no | yes | yes |
| Yang[248]: Design Verif. | 802.11 MAC | – | no | no | no | no | no | yes |
| Shono[220]: 802.11 SDR | 802.11 MAC | – | fully* | no | yes | no | no | part. |
| Arpinen[15]: Configurable | TUTWLAN | UML | fully* | no | no | yes | part. | yes |
| Dhar[47]: Integrated Devel. | 802.11 M+P | Click/gr | fully* | no | yes | no | part. | no |
| Lee[126]: Dual-Processor | 802.11 MAC | – | fully | no | no | no | part. | yes |
| Nychis: Enabling MAC[172] | 802.11 M+P | gr | fully | no | yes | no | part. | no |
| Hedde: MPSoC flex.[78] | 802.11 M+P | – | fully | no | no | yes | no | no |
| Tan: SORA[229] | 11abg M+P | – | fully | no | yes | n/a | no | part. |
| Nabi: Reconf. MAC[166] | 802.11 MAC | – | FPGA | no | yes | no | no | yes |
| Sankaran: Atheros 2x2 [206] | 11n P>M | n/a | n/a | yes | no | n/a | n/a | (area) |
| Wipro: Wild 802.11abg[245] | 11abg MAC | n/a | part. | yes | no | part. | n/a | no |
| Lantiq: 1x1 LiteMAC[121] | 11n M+P | – | part. | yes | no | no | n/a | yes |

M = MAC, P = PHY, n/a = not available or applicable, gr = gnu radio, (802.11) = generic *) but fail

### State-of-the-Art Summary

We aim to develop a WLAN MAC system that is commercially competitive while leveraging a systematic approach to the end of a flexible and generic system. As summarized in Table 2.3 for the most relevant pieces of related work, such an approach does not exist yet to the best of our knowledge:

- Often a generic *protocol version* of 802.11 is used, especially in SDR contexts. Sometimes, MAC and PHY layers (M+P) are considered, or the PHY layer is in the focus (P>M). Only commercial systems are known to fully implement 11n MACs, academical work is limited to 11a/g.
- Only few academic approaches support a high-level *programming model* (see also Sec. 2.2.3). However, they either fail to meet performance requirements or require the usual HW accelerators.
- *Flexibility* at the MAC layer was discussed in the previous sections: Few fully SW-based approaches exist, however, they fail other criteria such as supported protocol, programmability, and product relevance.
- No academic work could be found on the full MAC functionality relevant for *802.11n products*. For example, consideration of system dimensioning, precise memory requirements, different function splits, and host interface is typically missing. At the same

time, commercial systems are poorly described publicly and typically lack flexibility, generality, scalability, and a high-level programming model.

- Both *SDR* approaches and the support for multiple MAC protocols are becoming increasingly popular. However, none of the presented platforms was both systematically explored and applicable to 802.11n MAC products.
- Only few authors consider *systematic exploration* of the WLAN MAC layer and justify function splits, HW/SW trade-offs, and performance requirements.
- *Scalability* is only considered by few authors that, e.g., base their systems on MP-SoC platforms or explore advanced NoCs.
- *Quantitative results* typically concern chip area. Only few include performance or memory requirements. These results are compared to our solution in Section 6.8.4.

### 2.2.4 Future Trends and Challenges in Wireless (LAN)

Our platform concept must also be scalable to future versions of shared-medium and WLAN protocols. The success of the 802.11 standard so far was based on easy yet robust operation and over-provisioning of resources. However, more sophisticated approaches are necessary as channel and spectrum capacities reach their limit, traffic and QoS demands are increasing, and user requirements and deployment scenarios are evolving. These are partly addressed by **amendments** to 802.11 [79]:

- *IEEE 802.11k/T* – Radio resource measurement / performance prediction
- *IEEE 802.11s/z* – Mesh networking / direct link setup (DLS)
- *IEEE 802.11v* – Wireless network management
- *IEEE 802.11w* – Protected management frames
- *IEEE 802.11aa* – High-quality video transport streams
- *IEEE 802.11ac/ad* – Very High Throughput (VHT) at below / above 6 GHz

We project the following **challenges for future MACs** (in addition to Sec. 2.2.3) with differing impact on MAC performance and architecture requirements:

- *QoS and audio/video streaming (strong impact)* – The *11e* standard's EDCA is increasingly supported while central coordinators are not found in today's products. WLANs must reliably support VoIP. *11aa* targets time-synchronized, low-latency streaming services with graceful stream degradation.
- *MAC efficiency (strong impact)* – MAC efficiency is greatly improved with *11n*. In the future, inter-frame spaces and timing could be further optimized. Optionally, *Reverse Direction Grant (RDG)* aids bi-directional traffic such as VoIP and TCP, and *Direct Link Support (DLS, 11z)* conserves network resources. Unlike aggregation and timing, the impact of RDG and DLS on MAC architecture is limited.
- *Throughput (strong impact)* – In addition to the above, the operation of multiple independent MACs, i.e., multiplexing of time and space improves efficiency. Together broader spectrum bands (up to 100 MHz), better coding, and MIMO the throughput is projected to increase to 1 Gbit/s and beyond (*11ac*, *VHT*).

- *Link adaption and cognitive radio (strong impact)* – The available spectrum must be utilized more efficiently to increase reach (MIMO, *11y*) and interoperability by reducing interference (*11h*, *11v*). Important means to this end are the measurement standard (*11k*) and advanced dynamic link adaption techniques. Future devices may dynamically select the best available network, protocol, configuration, and channel, and utilize even small free gaps in the spectrum.
- *Versatile deployment(medium impact)* – Deployment, e.g., in automotive contexts (WAVE, *11p*) requires special consideration of network management, security, robustness, range, and low-latency operation [56]. Scalability is thus essential, and power is a major concern for mobile devices.
- *Security (limited impact)* – In addition to established standards (*11i*), encryption of management frames (*11w*) prevents denial-of-service and packet-insertion attacks.
- *Network management and integration (small impact)* – Many features will improve usability of wireless networks but have small impact on MAC architectures. For example, networks must be centrally managed in offices, or be easy to set up in homes. Others include user handoff and roaming (*11r*), mesh networks (*11s*), network management (*11v*), and interoperability in large networks (11u). For fast bridging (*11c*, *11s*), appropriate packet buffering must be provisioned.

A future need for an open, scalable, and flexible MAC layer can be inferred in conclusion. Even implementations of the 11n standard, although now finalized, are evolving, e.g., to properly utilize 40 MHz channels [237, 175] and features [137]. This is addressed by programmable systems that can be easily adapted and updated. Higher throughputs and low-latency operations to improve efficiency and QoS directly affect the MAC's architecture. The operation of multiple MACs at the same time requires scalable and concurrent systems. Furthermore, increasingly intelligent radios with frequency/space multiplexing will be based on complex SW and thus require powerful, programmable, and real-time capable platforms. Open architectures simplify systematic cross-layer design, measurements, and optimizations. With increasing true network convergence flexible MACs must complement SDR-based PHYs with multi-protocol operation. At the same time, systems must be cost-efficient and allow flexible function splits for better integration.

## 2.3 Relevant Device Configurations and Features

Following our goal to develop a scalable and flexible next-generation WLAN platform, we highlight relevant protocol features and define benchmark configurations. Trends from the preceding sections are reflected where possible and will be discussed in the light of our exploration results in Section 6.8.5. The selection of features that are relevant in terms of performance and architecture allows to model the reference application in Chapter 3 in a more focused way. Providing an indicative range of target configurations yields a scalable architecture and drives the architectural exploration of Chapter 6.

**WLAN features** differ in their impact on performance requirements and functional memory sizes, as listed in Table 2.4. Many features such as RDG and DLS only impact the management layer and are thus neglected for our exploration. At the same time, functions concerning aggregation, BlockACKs, and highly dynamic scheduling will be shown to have a strong impact on system architecture and memory hierarchy. Device dimensioning is reflected in

***Table 2.4*** *– WLAN features and their impact on* perf*ormance,* mem*ory, and* management.

| *Feature* | *Options** | | Impact** | *Explanation / Comment* |
|---|---|---|---|---|
| *TXOP continuation* | **yes** | no | perf | Multi-frame TXOP / contention-free burst (incr. eff.) |
| *Dynamic aggregation* | **yes** | no | perf | Dynamic adjustment of A-MPDUs to TXOP (incr. eff.) |
| *Imm. retransmission* | **yes** | no | perf | Direct response to BlockACK (incr. efficiency) |
| *BlockACK streams* | **4** | **48** | mem(+perf) | Extra buffering and scheduling required |
| *A-MPDU subframes* | **16** | **64** | perf(+mem) | Depends on aggregated packet sizes |
| *DLS* | yes | **no** | mgmt(+perf) | Direct Link Support / management, scheduling |
| *RDG* | yes | **no** | mgmt(+perf) | Reverse Direction Grant / management, scheduling |
| *Powersave (PSMP)* | yes | **no** | mgmt(+mem) | Requires extra queues for sleeping stations |
| *Link Adaptation* | **yes** | no | perf+mem | Fine grain control of rate and channel |
| *Supported STAs* | **16** | **64** | mem(+perf) | Access Point feature |
| *Number of MACs* | **1** | 4 | perf+mem | Improve network efficiency (Sec. 2.2.2) |

*) Considered choices are highlighted
**) Performance/memory requirements, parentheses indicate minor impact

***Table 2.5*** *– Summary of WLAN device reference configurations.*

| | 2x2 AP | 2x2 STA | 1x1 AP LiteMAC | 4x4 AP MultiMAC | VHT AP projected |
|---|---|---|---|---|---|
| *Device Type* | Full MAC | Full MAC | Lite-MAC | Full MAC | Full MAC |
| *Device Function* | Access Point | Station | Station | Access Point | Access Point |
| *Spatial Diversity* | 2x2 | 2x2 | 1x1 | 4x4 | e.g., 4x4 |
| *PHY Speed [Mbit/s]* | 300 | 300 | 150 | 600 | 1000 |
| *Multi-MAC* | – | – | – | 3x3 + 1x1 | e.g., 2x2 + 2x2 |
| *TXOP continuation* | yes | yes | no | yes | yes |
| *Dynamic aggregation* | yes | yes | no | yes | yes |
| *Supported STAs* | 32 | 1 | 16 | 64 | 64 |
| *BlockACK streams* | 16 | 4 | 8 | 32 | 48 |
| *Subframes per A-MPDU* | 16 | 16 | 16 | | – |

the benchmark configurations and more precisely analyzed – especially in terms of memory requirements – during the application analysis of Section 4.2.

This choice of features is probing the limits of 802.11n in terms of real-time performance and exceeds the state-of-the-art of today's systems. Maximum-sized A-MPDU aggregates, dynamic aggregation, TXOP planning, and immediate rescheduling after BlockACKs, however, maximize efficiency and throughput. In addition, efficient dynamic aggregation [137] and link adaption close to the PHY will become important in the future as discussed previously. Thus, our benchmark reflects future trends in terms of real-time performance of complex embedded protocol operations.

The **benchmark configurations** of Table 2.5 cover the complete range of current and future 11n devices. We focus on Full-MAC architectures, but also consider a Lite-MAC and thus require flexible function splits. Standalone systems and SoC integration with the host or the digital baseband are not in the focus of this thesis and only discussed in the context of the UMAC architecture (see Sec. 8.4 or, e.g., [207]). We can thus restrict further exploration to the given configurations and will target a single, universal MAC platform.

## 2.4 Chapter Conclusion

Following the technical introduction of IEEE 802.11 WLAN, an analysis of the **state of the art** and of future trends for wireless MAC architectures was presented. Extended MAC functionality – in addition to core functions – must be considered for comprehensive home-networking devices. Flexible function splits are needed between Full-MACs and Lite-MACs, which also impacts the memory architecture of the final device. No single approach from academia covers the full IEEE 802.11n MAC system function in a flexible and scalable architecture while offering a high-level programming model and exploring HW/SW trade-offs with quantitative results at the same time. Little information is available on commercial systems. However, we also expect a trend towards flexibility there, fostered by more complex protocols, time-to-market pressure, and reduced total cost of development and maintenance. In the future, high throughputs, increased efficiency, fine-grain cognitive radio control, and SDR will have a significant effect on MAC architectures. Thus, programmable, concurrent, real-time capable, and scalable platforms are required that offer a high-level programming model.

Indicative **device configurations** were identified that cover the complete range of 11n devices, including multi-MAC and future high-throughput systems. Features relevant in terms of performance and system architecture include most notably aggregation, QoS, and real-time channel access. These configurations and features will drive the development process in subsequent chapters:

- The *modeling* of a reference application in Chapter 3 can be focused on performance-relevant features and configurations. Benchmark scenarios are devised that reflect our device configurations.
- The architecture-independent *analysis* in Chapter 4 extends on the configurations, deriving memory and performance requirements from the reference application.
- Design *exploration* in Chapter 6 considers dynamic performance and memory requirements for the benchmarks. As a result, a scalable platform is developed and precise device costs for the proposed configurations are derived.

We will extend our findings in terms of architectural concepts and the usefulness of our approach towards the idea of a **universal MAC (UMAC)** throughout this thesis. Given the discussed MAC functionality, throughput and timing requirements, and standard complexity it can be concluded that Wireless LAN is at least representative if not a superset of other shared-medium home networking protocols. The importance of **productivity** in an industrial setting motivates a closer analysis of our approach. In this regard, a dedicated domain analysis can be considered a sensible and useful first step. Since the effort for this analysis, however, highly depends on the domain's complexity, novelty, and expertise of the designer(s), it is not quantified and assessed as a part of the development time. Subsequent development steps, however, will be assessed in greater detail, and the total efforts are summarized in the conclusion of Chapter 8.

# 3 IEEE 802.11 WLAN Reference Application

For step two of our methodology, a performance-indicative yet architecture-independent reference application is needed that captures essential system functions. Unfortunately, such a reference is not available for the IEEE 802.11n MAC layer. Therefore, we have developed an executable system model and its environment in a modular and domain specific language according to the configurations and features of Section 2.3.

Such a model allows performance benchmarking of MAC architectures and design space exploration. It serves as a starting point for efficient software-based implementations. In addition, it captures and documents protocol knowledge in an executable specification. It may serve as a golden model for test and verification. Finally, it can provide black-box function for network simulation and allows experimentation with protocol extensions.

After discussing modeling requirements for MAC-layer applications, a modeling language is selected and refined for deployment on resource-constraint network nodes. Then, a fully-functional WLAN system model implementation is presented – in its graphical syntax and with a detailed description of its main components and processing paths. Protocol timing, frame aggregation, and configuration require special attention. Based on the model, performance-indicative benchmark scenarios are defined and verified.

## 3.1 Modeling and Models of Wireless Protocols

### 3.1.1 Language and Model Requirements

Wireless LAN is a complex protocol that comprises a variety of functions and relies on exact timing (cf. Chap. 2). Thus, the full system function must be modeled. This is different from small-scale, homogeneous, and untimed algorithmic kernels that are often considered in literature. In addition, use cases and traffic environments are needed. This requires protocol instances to be configurable, e.g., in terms of their function, size, and mode of operation. For a **suitable modeling language** we therefore require:

- Modeling of the *full system function* at the MAC layer, especially of *timing-precise* behavior and *control flow*. This includes time-stamping and precise and dynamic scheduling. Packet and data flows must be coordinated. In a modular approach, this includes capturing dependencies between elements.
- *Domain-specific* concepts for packets and their processing. This includes packet generation, copying, and deletion, and special functions, e.g., for packet aggregation.
- *Appropriate abstractions* for fast modeling while focusing on essential aspects. The language must be as *expressive* as needed yet remain easy-to-use and *intuitive*.

- *Modular and extensible* models that encourage reuse. An existing element library for packet processing is beneficial. *Dynamic changes* must be considered, e.g., when the network topology changes.

Since our focus is on the device and its architecture rather than protocol performance, especially **performance-relevant aspects** must be captured precisely, and we require:

- *Executable models* that enable system validation and that serve as an executable specification. Models must be *performance indicative*, i.e., they must not fundamentally differ from the final implementation. A *path to implementation* is needed.
- Models to be *architecture independent*, enabling platform benchmarking and exploration. Models thus must expose *application behavior and requirements* and orthogonalize it from platform functions such as IO, scheduling, and memory management.
- *Data flow* must be explicit as it has a big architectural impact. This naturally captures inherent *parallelism* in packet flows and dependencies between elements. Similarly, *resource usage*, e.g., for storage must be explicit and finite.
- *Explicit communication* separated from computation, as is essential for mapping onto concurrent architectures. This means that state should be kept locally. Similarly, functions and structures must be provided at an appropriate *granularity*.
- *Modeling support*, as models quickly grow large and complex. This includes script integration and a graphical editor for generation and configuration.

### 3.1.2 Modeling Languages and Frameworks

A broad range of modeling approaches exists. For a comprehensive overview of approaches targeting protocol comparison and generic packet processing application, see [207]. We focus on the area of wireless systems and protocols as we discuss models of computation, frameworks, and languages.

**Models of Computation (MoCs)** are at the core of most modeling approaches [54]. They can be seen as design patterns that separate computation, communication, and control while addressing issues of concurrency and time. They describe the system behavior in an executable way as an interaction of components that guarantees desired system properties [95]. *Kahn Process Networks (KPNs)* and variants thereof are very popular. Introduced by Kahn [99], such models consist of sequential processes that run concurrently and communicate synchronously or via FIFOs. Even though they capture data flow [123], their behavior is untimed. *Discrete-Event (DE)* models follow a very general definition in that components communicate via events that are associated with a time-stamp. Most popular protocol modeling frameworks are based on DE, including *OMNeT++* [235] and *ns-2* [34]. However, these abstract the node architecture, focus on simulation at the network level, and often fail to provide realistic models of complex wireless behavior [72]. Other relevant MoCs include Petri Nets, Queuing Networks, and *Finite State Machines (FSMs)*. However, such models are often less intuitive and difficult to understand [207]. Similarly, model-driven approaches [73] are often overly formal [104] and cumbersome to use. A compiler-assisted approach deriving protocol implementations from declarative descriptions [136] focuses on behavioral and cooperative aspects.

From the evaluation of MAC architectures of Section 2.2.3 we deem the following **languages** most relevant in the context of MAC protocols and systems design:

- *SDL* – The *Specification and Description Language* is popular for wireless system design (e.g., [179, 75]). It is based on both FSMs and KPNs. However, data-flow, computation, and timing are not first-class citizens. Implementations are rather inefficient. In addition, models can grow overly complicated, a possible reason why the SDL description of IEEE 802.11a has been discontinued despite its popularity.
- *UML* – The *Unified Modeling Language* is a general purpose language. Profiles exist for embedded systems that are actively used. However, the focus is put on structural aspects and specification, and no efficient implementation of a wireless system has been published to date [15].
- *gnu radio* – This data-flow language based on C++ is popular for implementing the PHY layer in SDRs. Some authors try to extend the scope to the MAC layer, but report lack of expressiveness, e.g. for timed behavior and state machines [47].
- *Click* – is a domain-specific, declarative language for implementing packet processing applications. It has been found suitable for modeling the MAC layer [47, 207] and will thus be discussed further in Section 3.2.

### 3.1.3 Models of Wireless LAN Systems

To the best of our knowledge, no executable reference or full system model is publicly available for the IEEE 802.11n MAC layer. A reference for IEEE 802.11 in SDL as part of the standard document has been discontinued. Only test specifications such as IEEE 802.11T exist that describe a host of traffic patterns and metrics [90]. Models for IEEE 802.11g in Click are discussed in [47] and available with the Click framework [230]. Unfortunately, no detailed description is available for the former, and the latter only covers management functionality. Still, it can partly be reused for our purposes in Section 3.3.3.

Other existing models either target legacy protocols, focus on isolated aspects, or fall short of satisfying our requirements. A number of WLAN models exist, e.g., in the popular OmNET++ and ns-2 frameworks that abstract performance, lack detail, and have not yielded efficient implementations. The legacy protocol is modeled, e.g., in [190, 244]. Recent ns-2 models covering the frame aggregation facet of 802.11n include [135, 238, 221]. Often, isolated aspects such as aggregation [167, 141] or EDCA [92] are examined using analytical models. Kukkala [115] models the legacy standard in UML.

## 3.2 Modeling with Click

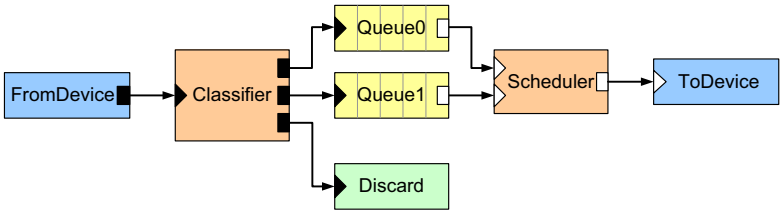
We find *Click* most suitable for our requirements (Sec. 3.1.1) as outlined in the following. However, the special characteristics of WLAN to be deployed on resource-constraint devices necessitate modeling techniques and extensions as described in Section 3.2.2.

### 3.2.1 The Click Modeling Language

Click [113] is a widely-used framework for packet processing applications based on a domain-specific, declarative language. The application library includes standard functions and few higher-layer wireless modules. Using Click, functionally correct and executable

models can be derived quickly. Click has also been used in network simulation, e.g., with ns-2 [170]. Further information is found in [112] and the Click web site [230].

As described by Sauer [207], **Click applications** are composed from elements that are linked by directed connections. The elements describe common computational operations whereas connections specify the flow of packets between elements. Packets are the only data type that can be communicated; their generic format supports arbitrary communication protocols. All application state is kept locally within elements. Two packet communication patterns are distinguished in Click: *push* and *pull*. A push is initiated by a packet source, modeling the arrival of packets. Pulls are initiated by a packet sink, modeling available space in an outbound resource. Figure 3.1 gives an example for Click's graphical and textual syntax. Packets enter the system by the *FromDevice* element (push output [black]). The packets flow to the input of a Classifier. This element forwards packets depending on the result of internal processing, e.g., filtering header fields. Two of its outputs are connected to queues. In Click, queues are explicit and have push inputs and pull outputs (white). Hence, the *ToDevice* can pull packets out of the queues at its own rate, removing them from the system. Pulling of packets happens via the Scheduler, which selects a packet from its inputs depending on its policy (e.g., priority).




```
FromDevice -> cl::Classifier [0] -> q0::Queue -> [0] s::Scheduler -> ToDevice;
                                 cl [1] -> q1::Queue -> [1] s;
                                 cl [2] -> Discard;
```

***Figure 3.1*** – *Click example: graphical and textual representations.*

The **Click language** is a graphical and textual coordination language to instantiate and configure elements and to describe the packet flow between them.[1] Compound elements enable hierarchy. Single elements are implemented in C/C++ and stored in a library. Connections are mapped to method bindings. For push connections, the thread of execution flows with the data, i.e. the target's push method is called with the packet as parameter. For pull connections, the generation of a packet is triggered by calling the pull method of an upstream element with the packet being the return value. This pattern directly results in an efficient implementation. Click's notion of time is explicit and metric. Packets in Click can carry a time stamp. Elements may use the time stamp and access system time.

Click satisfies most **requirements** of Section 3.1.1 (see also [207]). It is domain-specific, sufficiently expressive and intuitive, and encourages efficient implementations. Since elements describe a particular function, encapsulate all state, and have a uniform interface, reuse and modularity are natural. Click has been found especially suitable for modeling the MAC layer, e.g, by Dhar [47]: It has concepts of time and packets, allows coordination between transmit and receive flows, supports complex scheduling, and both local and global state

[1] Click defines a model of computation, see, e.g., Lee [122] or our discussion in Section 5.1.2.

can be maintained. Its applicability to performance evaluation and target implementation are shown in subsequent chapters. Click also has shortcomings in terms of timing and communication within the model that are discussed in the next section. The problem of reconfigurability is discussed in Section 3.3.6.

### 3.2.2 Modeling Techniques and Click Extensions

The original Click framework targets router implementations running on single-processor Linux systems. In order to adapt it to MAC-layer modeling for embedded multiprocessors, we address issues regarding modeling, timing and communication within our own framework – SystemClick (see Sec. 5.2.4).

**Modeling.** It is imperative to find "good" granularities and function splits for Click elements. If too much is implemented in a single element, the model's modularity is lost. It is thus harder to understand, to partition, and to exploit its parallelism. Using only elements with very limited functionality, however, makes models complex and may severely impact performance. Similarly, a good trade-off has to be found between generic and specialized elements: Universal elements would, e.g., classify a packet internally whereas a specialized element would rely on classification already being done elsewhere.

**Timing.** In the Click framework, execution times of tasks dependent on the host system, and scheduling is based on inexact means provided by Linux. Thus, we address:

- *Simulation* – Processing times are abstracted for modeling by a fixed value small enough not to interfere with protocol operation. This ensures event causality in the model. Unlike ideal discrete event simulation (e.g., in *ns-2* [34]), this is more realistic regarding the target system and opens the path to implementation.
- *Scheduling* – MAC applications are timed precisely relative to external events such as packet reception. As system time progresses during execution, relative scheduling alone is insufficient. We introduce a *ScheduleAt* directive that enables precise scheduling based on, e.g., time stamps annotated to packets. Scheduled times in the past indicate problems in the application logic and are reported to the user.

**Communication** within a Click model refers to the interaction of elements. Natural to Click is the flow of packets along push/pull connections. However, we identify additional schemes that cannot be modeled properly in Click:

- A communication means outside the flow of packets is needed, e.g., when a packet is forwarded to the host but other elements need to be notified about its reception afterwards. We refer to this as active *one-to-one* communication.
- Similarly, when many elements need to be actively informed about, e.g., a state change, we have an *active one-to-many* communication or broadcast.
- Distinct pieces of information such as global configuration need to be accessed by many elements. Broadcasting this information actively to all subscribers, however, is too expensive. In this case, we prefer a *passive one-to-many* communication.
- Packets have annotations to transport additional *sideband* information. This works well in downstream direction along push processing chains. When packets are pulled, however, the "caller" cannot pass parameters to the upstream elements

- In a demand-driven setup with multiple producers and one consumer, the consumer cannot pull from the right producer. This is a *many-to-one* communication.

**Communication Extensions.** Within our approach, we introduce the following artifacts and extensions to address above mentioned communication patterns:

- *Tokens* – provide an efficient means for active one-to-one communication and transfer the control flow in Click. Based on Click packets, tokens are distinct and provide a semantic and data structure for non-packet data types such as boolean. Tokens reuse standard Click ports and connections, but are assumed to be unique or maintain a reference count, which makes it unnecessary to duplicate or kill them. Relying on the *Tee* and *Classifier* element infrastructure of Click active one-to-many communication can be efficiently implemented as well.

- *Shared data objects* – represent a single data object shared among several elements. It makes data dependencies explicit, allowing to cope with it during mapping. Shared objects are a means for one-to-one communication along pull chains and passive one-to-many communications. If the shared data are complex, the object may be structured and modifiable only through access methods (cf. Sec. 3.3.6).

- *Annotation with direct access* – may be necessary to directly access an element via a reference in a many-to-one situation in order to limit model complexity. A reference to the producer is annotated to the packet and the consumer directly accesses the producer through a function call.

In summary, our findings and extensions address the shortcomings of Click with respect to modeling complex, time-dependent applications on resource-constraint systems. They help to aim the system model better towards the final implementation on the target device, while still remaining platform independent.

## 3.3 WLAN Model

The model described in the following captures the full IEEE 802.11 WLAN system function in Click, including QoS and high-throughput extensions (11n). Model setups comprise instances of AP and STAs combined with packet sources and a shared medium. The model itself is modular and comprises interconnected processing paths. Essential aspects of its implementation include protocol timing, aggregation, and model reconfiguration.

### 3.3.1 IEEE 802.11n Protocol Function

The model captures the protocol timing and state of both the legacy IEEE 802.11abg and IEEE 802.11e/n MAC layer. We focus on **performance-relevant aspects** and implement the following features:

- In *legacy mode*, fragmentation and RTS/CTS protection can be used. Medium access is based on a contention window, and frames are acknowledged and re-sent one by one. On collisions, a back-off mechanism increases the contention window.

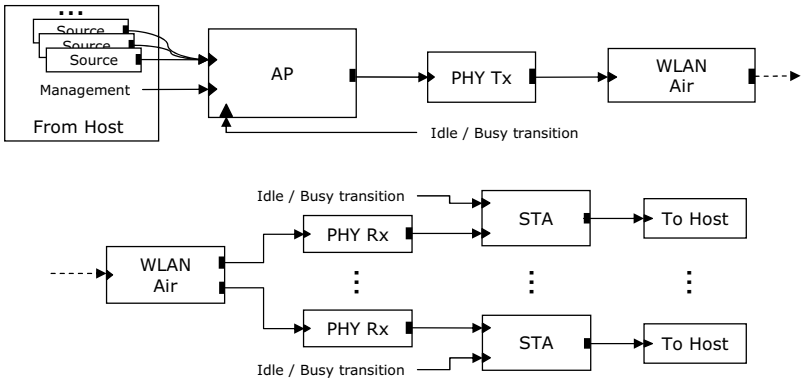



***Figure 3.2** – Generic setup for the transmission (upper part) and reception path (lower part)*

- *QoS extensions (11e)* introduce four prioritized access categories (AC) for medium access. Stations acquire TXOPs per AC, e.g., by completing an RTS/CTS exchange and subsequently send packets. Acknowledgment of frames is done in blocks (Immediate BlockACK) with timely retransmission of failed packets.
- *HT extensions (11n)* introduce two mandatory aggregation schemes: The aggregation of service data units (A-MSDU) relieves per-frame MAC processing. Aggregation at the PHY layer (A-MPDU) reduces overhead by concatenating protected subframes. Total length must match the current TXOP.

In addition, **management functions** such as beacons, station authentication and association as well as the exchange of configuration data are comprised in the model. Although operational and functionally correct, WEP security and rate selection modules are merely placeholders for more sophisticated schemes imposing similar loads on a system.

Features that are not performance relevant (cf. Sec. 2.3) have been abstracted, including detailed PHY handling and higher-layer management.[2] Especially *HT Control* mechanisms (training, sounding, beam-forming) and recommended MCS are not supported. Channel management, *Direct Link Support (DLS)*, *Reverse Direction Grant (RDG)* and contention-free operation modes such as PSMP are in principle covered by the model but would simply add more management overhead. All other features can be omitted as they refer to higher management functions only, such as negotiation of BlockACK streams.

### 3.3.2 Setup and Interfaces

The model setups include one *Access Point (AP)* and several *Stations (STAs)*. Figure 3.2 provides a generic overview of the setup and the flow of packets. Packets are generated by packet sources, e.g., providing IMIX [13], VoIP, or HDTV traffic. The abstracted host then pushes Ethernet-encapsulated packets into the system. The WLAN standard defines this interface as the *Service Access Point (SAP)* of the MAC layer. Packets are processed by the MAC and forwarded to the PHY interface. Eventually, received packets are again processed, Ethernet-encapsulated, and forwarded to the host.

[2]However, such features must be added as needed for prototype implementation in Chapter 7.

The AP and the STAs have additional inputs. The PHY generates information tokens indicating the medium state and the end of frame transmissions. The host provides high-level management frames and triggers management functions implemented in the MAC model such as authentication and association (cf. Sec. 3.3.6). Our model uses special (compound) elements for the PHY layer and the shared medium:

- The *PHY TX* element queues aggregated packets and delays transmission according to an annotated time stamp.
- The *WLAN Air* element generates both timing behavior of frame transmissions and half-duplex busy signaling. Frames aggregated at the PHY layer are transmitted back to back. Bit errors can be introduced at random or in case collisions occur. It broadcasts received packets to all RX PHYs.
- The *PHY RX* element receives and time stamps the packets from the air and transmits them to the AP/STAs.

### 3.3.3 Model Implementation

The Click model for IEEE 802.11n is shown in its graphical representation in Figure 3.3. Processing paths for transmission (TX) are found in the left-hand part and for reception (RX) in the right-hand part. Time-critical functions are found close to the air side (bottom), whereas pre-/postprocessing stages are close to the host (top). This relates to the core and extended functions of Section 2.2.1. The figure exemplary features three TX/RX packet stream processing paths (*TID streams*) in different configurations for Access Category 0 (AC 0), and the legacy path. Other ACs are present in the model but not shown. All processing paths can be configured by a ReconfigurationManager, which writes a global StationInfoBase every time the connection state changes (as detailed in Sec. 3.3.6).

**WLAN Click Elements.** Standard Click elements provide a good infrastructure for our implementation of the IEEE 802.11n protocol function. Some of Click's higher-layer WLAN management functions can be reused and have been extended as needed. Still, the library had to be extended substantially, e.g., for real-time and 11n processing:[3]

- *Protocol timing* is handled by the *EDCA* element, which schedules packets and sequences for different ACs. The *Update_NAV* element maintains a counter for virtual carrier sensing and communicates its busy/idle state using tokens.
- *Response frames* are generated by the *GenACK* and *GenCTS* elements for incoming unicast data frames and received RTS frames, respectively. Both elements extract the necessary information for the generation from the received frames.
- *Aggregation* of A-MSDU frames is done by joining Click packets until a limit or timeout is reached (*Wifi_Aggregate_MSDU_AP*). For A-MPDU, first the aggregated packet length is determined (*Wifi_AMPDU_Initial*) before packets are removed from the buffer and subframe headers are inserted (*Wifi_AMPDU_Continue*).
- *Block Acknowledgments* are generated by the *WifiReorderBuffer*, which keeps track of received frames. At the sender, BlockACKs are preprocessed by the *WifiBlockAckResponder* before the *WifiReplayBuffer* schedules lost frames for retransmission.

[3] The application library has been implemented in or ported to C within our framework SystemClick (cf. Sec. 5.2.4) for increased efficiency and to ease model deployment on resource-constraint systems.

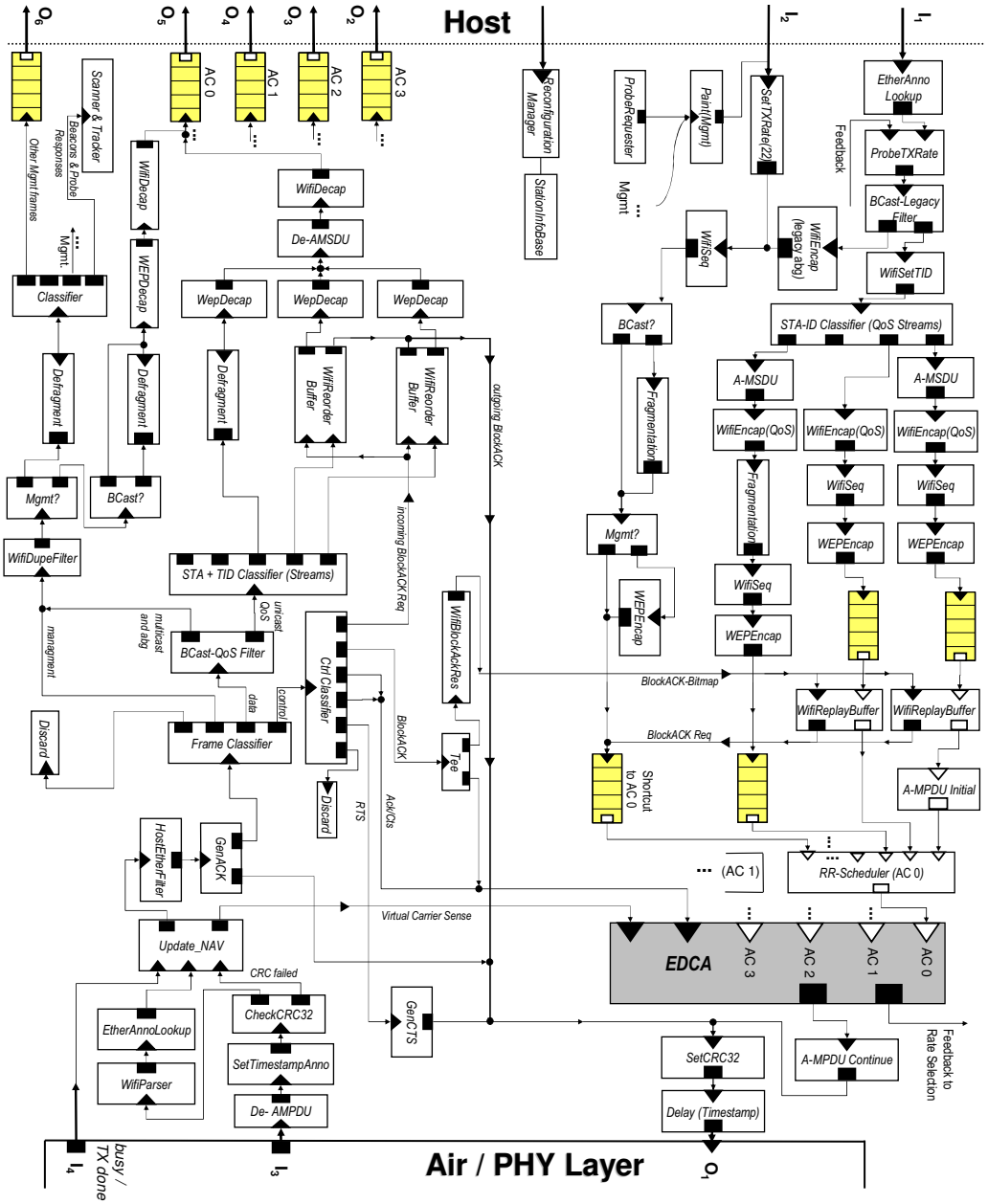



***Figure 3.3*** *– Click model of a WLAN 802.11abg/en MAC with transmit paths (TX) on the left-hand side and receive paths (RX) on the right-hand side.*

**Processing Paths.** The main receive and transmit packet processing paths through the Click model as shown in Figure 3.3 are detailed in subsequent paragraphs. A number of interconnections in between these main TX and RX paths exist (cf. Fig. 3.3). This reflects the need for immediate generation of responses to incoming frames, which is characteristic of shared-medium protocols. RTS frames need to be responded by a CTS message, and incoming packets with single acknowledgment policy must trigger an immediate ACK

response. This is more complex in the case of a *Block Acknowledgment (BlockACK)*, which is processed by the replay buffer. All incoming control frames are forwarded to the EDCA element as they usually affect the scheduling of the next transaction. Incoming BlockACK requests can trigger the immediate generation of a BlockACK frame at the ReorderBuffer. Information about successfully transmitted or failed packets is provided as feedback to the rate selection algorithm (not shown in the figure).

**A.1) Outbound 11abg data frames.** All frames from the host (input $I_1$) pass rate selection and get a STA-ID based on Ethernet address and TID. Legacy (11abg) frames are sorted out, encapsulated, receive a sequence number, and are fragmented/encrypted as needed. Stored in the best effort traffic queue, they are pulled round-robbin by the EDCA. The EDCA sets the duration field, TX time, and schedules RTS/CTS protection if indicated. Eventually, every frame is CRC'ed and sent as specified by an annotated time stamp to the PHY ($O_2$).

**A.2-4) Outbound 11n data frames.** 11n frames are classified into QoS paths and have additional header fields and TID set. If configured, aggregation into A-MSDUs and encryption are performed before being queued. The EDCA schedules and pulls frames for a given AC. In BlockACK mode, the WifiReplayBuffer tracks frames, replays them if not acknowledged, and issues BlockACK requests. Depending on the TXOP length, max.-sized A-MPDUs are initially prepared by computing the aggregate length. The EDCA performs RTS/CTS protection to start TXOPs and sends CF-END frames when no more packets are available. Finally, the actual aggregation is done and CRC-protected frames are sent as scheduled.

**B.1) Inbound 11abg data frames.** Packets from the PHY ($I_3$) are timestamped and CRC checked. They are annotated with STA-ID, TID and type. The NAV timer is updated with the frames' duration. The model's state is also updated if CRC checks fail or if medium state changes ($I_4$). Next, frames for other networks are discarded and unicast frames are acknowledged. After classification and duplicate removal, frames are decrypted and reassembled. After header translation to Ethernet, frames are forwarded to the host ($O_6$).

**B.2-4) Inbound 11n data frames.** A-MPDUs are deaggregated, member frames are marked and forwarded subsequently the same as 11abg frames. STA-ID classification then forwards the frames into the proper QoS path. The WifiReorderBuffer keeps track of received frames and releases them in correct sequence. Frames are then decrypted and A-MSDUs de-aggregated into single frames before being forwarded to the host ($O_{2-5}$).

**C.1) Outbound 11abg acknowledgment.** ACK frames are generated upon reception of a valid abg unicast data frame or if indicated by the ACK policy. The frame is scheduled after SIFS, forwarded to CRC protection, and sent out.

**C.2) Outbound 11n BlockACK.** BlockACKs are triggered by explicit or implicit requests and generated by the WifiReorderBuffer. Explicit requests are treated as inbound control frames. Responses are scheduled after SIFS. The implicit mechanism extracts the trigger from A-MPDUs and schedules the response, either after the last member is processed or if time to send runs out.

**D.1) Inbound 11abg acknowledgment.** Inbound ACK frames are processed as in (B.1), but classified as control frames and forwarded to EDCA. There, the stored frame is killed and the next transmission can proceed. A timeout in the EDCA detects failed transmissions and causes retransmissions according to retry counters. Feedback is given to rate selection.

**D.2) Inbound 11n BlockACK.** Inbound BlockACKs are processed as in (B.1), but classified as 11n control frames and forwarded to the WifiBlockAckResponder. The extracted bitmap is then forwarded to the proper WifiReplayBuffer(s). There, acknowledged frames are cleared, and lost frames prepared for retransmission. A timeout detects missing BlockACKs.

**E) RTS/CTS.** If an outbound frame exceeds the RTS threshold or a TXOP is to be used, EDCA issues an RTS prior to actual transmission. At the receiver of an RTS, a CTS is generated similar to ACK

frame processing (C.1/D.1). Completing the exchange, inbound CTS are classified and trigger the transmission of the outbound frame after SIFS in the EDCA.

**F) Management Frames.** Management frames are generated by the model or can be provided by the host through dedicated ports ($I_2$). Outbound management frames are forwarded to the legacy path (B.1) without additional Wifi encapsulation. Incoming management frames are received like data frames, but assorted by classification and either directly processed or forwarded to the host.

### 3.3.4 Implementation Essentials

Click's run-to-completion model of computation and its scheduling scheme cannot guarantee **precise timing** in the presence of processing delays. Thus, we introduce a scheduling scheme based on time stamps. Incoming packets and the CCA are stamped upon reception to have a consistent timing reference throughout their processing. Tasks are scheduled relative to this time stamp. For transmission, packets are annotated with the scheduled transmit time and delayed accordingly.

Central to the model's timing is the **EDCA element** that schedules sequences. It maintains backoff counters per AC and pulls packets from schedulers that select among all streams of a given AC. The backoff management is realized with two auxiliary functions: `findNextTriggerTime()` initially computes the backoff for the AC based on the current backoff situation and the given parameters (e.g., aSlotTime). `updateBackoffTimers()` is used to update backoff counters when the medium went busy during the contention process. The EDCA element is implemented as a state machine with states presented in Table 3.1 and two different transition functions, `RUN` and `PUSH`:

- The `RUN` routine is triggered by timers: When the EDCA is in a waiting state, every 9 $\mu$s (aSlotTime) the queues are pulled for new packets. Packets are then parsed to determine the next expected state as well as packet duration, sequence timeout, and possible retransmission. In the contention process, the `getNextAction()` function is used to update states and to schedule packet sequences if necessary.

- The `PUSH` routine is activated by feedback packets. If indicated, the EDCA schedules the next packet for transmission. In two cases, the `getNextAction()` function is called: A timeout is reached while waiting for a response. Or, no more transmission packets are left within a TXOP and a CF-END packet must be sent.

***Table 3.1*** *– States of the EDCA element's state machine.*

| State | Description |
|---|---|
| Wait for timer | The EDCA waits to check if new packets are available for transmission. |
| Wait for CTS | After sending a RTS, the EDCA waits for a CTS. |
| Wait for ACK | After sending a QoS frame, the EDCA waits for an ACK. |
| Wait for BlockAck | After sending a BlockAck request, the EDCA waits for a BlockAck. |
| Wait for end of TXOP | The EDCA is in a TXOP but has no more packets to send. |
| Wait for packet to finish | A timeout occured during transmission but the medium is still busy. |
| Wait for CTS finish | A CTS timeout occurred but the medium is still busy. |
| Wait for ACK finish | An ACK timeout occurred but the medium is still busy. |
| Wait for BlockAck finish | A BlockACK timeout occurred but the medium is still busy. |
| Packet scheduled | The EDCA has scheduled a packet for transmission. |

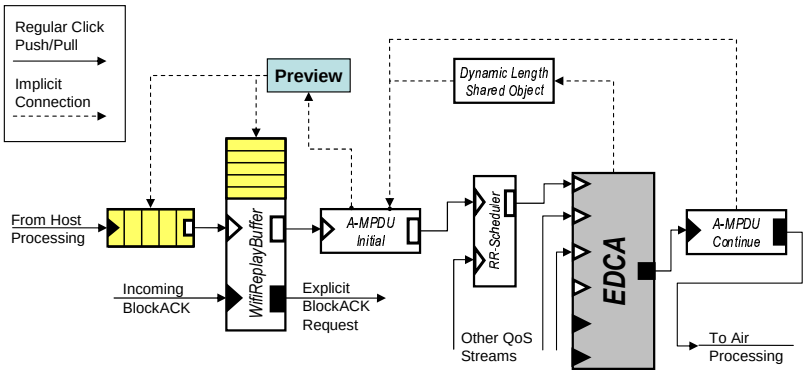



***Figure 3.4*** *– Implementation of the* ReplayBuffer *part of A-MPDU aggregation in Click.*

The EDCA element has precisely timed transitions at timeouts and when the virtual carrier sense changes. It is notified about relevant busy-idle or idle-busy medium transition by the *Update_NAV* element, which maintains a NAV counter based on the duration field of incoming packets. Knowing the medium state, the EDCA is able to defer timeouts if a packet reception is still in process.

The **A-MPDU aggregation process** is implemented by the *ReplayBuffer* and the *A-MPDU* aggregators (cf. Fig. 3.4). The ReplayBuffer keeps track of transmitted frames. When it reaches a limit, a BlockAck request is triggered. From incoming BlockAcks, a list of lost frames is extracted and only those are kept for retransmission. Anticipating an optimized implementation, aggregation has been split into two parts to decrease processing in the critical scheduling path: When the EDCA pulls for packets, the *A-MPDU_Initial* element first determines the possible aggregation length from a shared object. Then, queues with new or lost frames are previewed directly and a boiler-plate packet is generated. This packet is a place holder for scheduling and contains the number of frames to aggregate, the total length, and a reference to the ReplayBuffer that stores the remaining frames. Finally, *A-MPDU_Continue* performs the actual aggregation, pulling and processing follow-up packets off the indicated ReplayBuffer (cf. communication patterns in Sec. 3.2.2).

It is important to note that A-MPDU in our application *dynamically* adjusts the aggregate's length to the remaining TXOP time. This introduces more complex processing. While many state-of-the-art products perform aggregation statically in advance, the dynamic approach is vital for maximum throughput. This is especially true when dynamic effects during a TXOP – such as packet loss or RDG – make a static schedule impossible.

### 3.3.5 Model Configuration Parameters

Both the model structure and the protocol behavior can be customized. For the purpose of this thesis, all structural parameters are static whereas protocol settings can be changed at runtime (cf. Sec. 3.3.6). The model's **structure** is influenced by the following parameters:

- *Number and type of MACs* – Includes the supported IEEE 802.11 standard (Legacy, QoS, QoS+HT). In addition, multiple MACs can be instantiated, e.g., in order to support legacy operation and 802.11n at the same time.

- *Number of supported stations and TIDs* – The AP supports up to 32 stations. The *Traffic Identifiers (TIDs)* represent user priorities or traffic streams. Each TID for each station has a unique processing path that can be configured independently and includes aggregation functions and queuing.

The following parameters affect the model's **behavior** at runtime and can be set, e.g., via a global configuration database:

- *EDCA operation* depends on, e.g.: TXOP length, AIFSN, cw_min and cw_max that determine the contention window size, slot timing, retry counters, and thresholds.
- The total number of TIDs with *aggregated BlockACK streams* as well as the maximum size and maximum number of packets in A-MPDUs can be limited.
- Other parameters include the supported *rate* set, *timeouts* (e.g., for A-MSDU aggregation and retransmission of BlockACK Requests), and *encryption* schemes.
- The *network topology* can also be changed. This includes STA instances and characteristics of the medium such as the packet drop rate and the type of bit errors.

### 3.3.6 Management Functions and Model Reconfiguration

Changes in the network environment, management functions, and user interaction necessitate dynamic model reconfiguration. Wireless stations with unique capabilities join and leave the BSS. Connected stations add or remove streams with different aggregation types and acknowledgment policies. Higher management layers change link and QoS settings or may retrieve cross-layer information. We introduce an interface for such operations, discuss reconfiguration requirements and choices, and explain a solution chosen for the current model by giving an example.

High-level control of a MAC is done by a management entity through a **management interface**. Depending on the function split, this entity can be located on the same device or communicate through a remote interface. It must be possible to choose the function split according to the needs of the target system. Our model includes exemplary management functionality and provides an interface to external management functions, e.g., implemented as part of a Linux protocol stack. This interface is based on (Click) trigger packets and basically serves two purposes: It can provide user stimulus to management functions implemented inside the model (e.g., trigger association). And, it conveys parameter changes from external entities (e.g., to set up a BlockACK stream).

From the parameters of Section 3.3.6, different requirements become apparent that must be fulfilled by mechanisms that guarantee consistency and avoid packet loss:

- Changes to *single parameters in centralized elements*, e.g., network name or fragmentation threshold. Changes to queue and buffer sizes may require reallocation.
- Addition or removal of TID *streams*, e.g., by changing packet classification or graph structure. In addition, streams must be configured, e.g., to toggle aggregation.

**(Re-)configuration mechanisms.** Click uses a handler interface based on the Linux file system for run-time graph reconfiguration. It is utilized, e.g., by XORP [74] for a router control plane or in OpenFlow [154]. However, special consideration is necessary since we target a deeply embedded multiprocessor. Local, i.e., distributed configuration should

consequently be preferred to a central database and shared memory solutions that may become bottlenecks. In addition, memory resources are extremely limited.

Especially, multiple traffic streams must be modeled carefully. Two main approaches exist in modular languages (as also discussed in a different context in [101]): (1) Splitting the streams explicitly into different paths, e.g., per station or TID. Or (2), using a single path where every element classifies and processes incoming packets accordingly. Each approach offers different trade-offs: Explicit modeling incurs memory overhead from multiple element instances. Reconfiguration necessitates changes to the graph itself. However, configuration is exported into the model graph, thus making it amendable to analysis, mapping, and optimization. On the other hand, using a single path, computational overhead is incurred for classification at every processing stage. Streams and their configuration is not visible at the graph level and elements are more complex. Therefore, reconfiguration is limited to state kept inside single elements.

Our model combines these approaches but keeps TID streams explicit to ease mapping later in the design process. Thus, methods are needed to **(re-)configure traffic streams**:

- In a *fully static setup*, features are configured within elements only. A central configuration database is accessed at runtime by every element in *passive one-to-many* communication, or side-band annotation can be used (cf. Sec. 3.2.2). Alternatively, *active reconfiguration* saves runtime overhead, but elements must be addressable regardless of their mapping by a global directory similar to Click's handlers.
- Reducing memory usage, a *partially static setup* with a fixed number of streams can be allocated statically and reassigned as needed. This requires management of free streams and either *rewiring* of connections or reconfiguration of classifiers.
- *Complete graph reconfiguration* is memory-efficient, but necessitates dynamic generation, deletion, and rewiring. Individual elements need to be addressed and configured appropriately, and graph changes must be modeled appropriately, e.g., by subgraphs or in a reconfiguration language.

Our initial model implementation comprised a static setup, but the partially static setup proved to be a better solution during prototypical implementation. In fact, we can assume a globally limited number of streams for the memory requirement analysis of Chapter 4. For further discussion of reconfiguration, the reader is referred to literature: Liu [141] has proposed a hashtable-based approach to reassign TID traffic streams for WLAN, similar to our partial setup. Targeting configuration of real-time SW, Rasche [193] leverages algebraic category theory to describe both computation and parameters. Graph reconfiguration[4] is overly complex, and no pragmatic, light-weight solution to convey parameter and structural changes to a distributed graph has been established so far.

Configuration for associated stations including settings for TID streams is stored in a central lookup facility – the ***Station Info Base (SIB)***. Entries for connected stations are identified with a unique *Station ID (STA-ID)* and support a number of TIDs. STA-ID/TID are thus required for lookups from stream elements. These values can either be configured statically or be learned from packet annotations. Entries are accessed, added, and removed at runtime

[4] For example, Balani [17] presents a large-scale framework for in-situ reconfiguration based on a run-time wiring engine. In the context of active networks, Kanada [101] compares element connections, namely direct port-based and implicit label-based. It is shown that implicit graphs are easier to configure, but direct connections are better suited for distributed parallel implementations. Neuendoerffer [168] presents an abstract model for reconfiguration data-flow graphs based on hierarchical decomposition.

***Table 3.2*** *– System benchmark scenarios.*

| *Name* | *Description* |
|---|---|
| HT | Unidirectional high throughput (HT) setup between an access point (AP) and a single station (STA) comprising packets of different size distributions (min, max, typ, IMIX). |
| QoS | Typical home networking setup with AP and four STAs comprising all applications of Table 3.3 (/w file) to expose queuing and channel access timing. Additional VoIP, VC, HDTV and background STAs may be added (scenarios QoS2, QoS3, etc.). |
| Cmplx | AP and 32 STAs with IMIX traffic (1Mbps) for high model complexity exposing memory and configuration bounds. |
| (11abg) | Based on a previous implementation of IEEE 802.11abg this scenario is only used as a reference. |
| (11e) | Simplified configuration for IEEE 802.11e only supporting QoS extensions with 4 ACs. |

by local wrapper functions, and the SIB is mapped to shared memory. The SIB implements passive one-to-many communication (cf. Sec. 3.2.2), i.e., it is polled by elements. A *dirty* field is incremented upon updates to the SIB to indicate changes. It can be cached locally to reduce shared memory accesses.

## 3.4 System Benchmark Scenarios

Leveraging the WLAN system function as described in preceding sections, benchmark scenarios can now be defined that reflect and refine the reference device configurations of Section 2.3. This will enable further analysis and indicative exploration. Three **essential facets** must be addressed by such specifications for reliable and comparable results [231]:

- *System function* – unambiguously specifies the function of the system in form of a functionally correct and executable model.
- *Environment* – defines the surroundings of the system, its configuration, and parameters. This includes use cases and traffic setup.
- *Measurement* – describes how performance and the quality of results are evaluated.

Environment and measurement must now be defined in addition to precise configuration of the system function. For this purpose, the existing IEEE 802.11T specification describes 37 use cases, 19 usage models, and 24 different applications. But such a multitude is not necessary for the evaluation of different MAC architectures.

Instead, we propose three **base scenarios** that stress a MAC architecture's performance in terms of scheduling, high-throughput, and complex setup (cf. Tab. 3.2). The scenarios instantiate MAC system functions as well as host environments (traffic generators, packet sinks, management triggers), protocol checkers, and a combined air/PHY channel module (cf. Sec. 3.3.2). All stations are configured for 802.11n mixed mode, and a channel of 300 Mbit/s with standard timing is assumed: slot time 9 $\mu$s, SIFS 16 $\mu$s, PHY preamble and header ($24 + 4 * \text{nr\_mimo\_streams}$) $\mu$s, and CCA 4 $\mu$s. Unless stated otherwise, a constant error rate of 5 % for frames and A-MPDU subframes is assumed. The **measurement** facet is addressed in the following and further detailed during the exploration step, based on the response time requirements of Section 4.1.2.

Traffic is generated by the QoS applications of Table 3.3, and other model parameters are listed in Table 3.4. QoS applications are associated with traffic identifiers (TIDs) and mapped directly onto ACs with given parameters for TXOPs and aggregation. We depart from

***Table 3.3*** *– Benchmark applications and their characteristics.*

| *Application* | *Throughput* | *Pkt. Size* | *Delay* | *TID* |
|---|---|---|---|---|
| VoIP* | 0.096 Mbit/s | 120 B | 30 ms | 3 |
| Video Conf* | 2 Mbit/s | 512 B | 100 ms | 2 |
| HDTV | 24 Mbit/s | 1500 B | 200 ms | 1 |
| Internet/File(*) | 1/150 Mbit/s | IMIX[1] | BE[2] | 0 |

* Two-way communication 1) Internet IMIX distribution 2) Best Effort

***Table 3.4*** *– Settings for QoS and aggregation parameters.*

| *AC/TID* | *TXOP* | *A-MSDU* max. size | *A-MSDU* timeout | *A-MPDU* max. size |
|---|---|---|---|---|
| 0 | 1.5 ms | 4096 B | 50 ms | 64 Pkts. |
| 1 | 3 ms | 4096 B | 30 ms | 32 Pkts. |
| 2 | 0 | 2300 B | 10 ms | - |
| 3 | 0 | - | - | - |

***Table 3.5*** *– Scaling the QoS setup for different PHY bandwidths.*

| – | 1x1 | 2x2 | 3x3 | 4x4 | VHT |
|---|---|---|---|---|---|
| Channel Capacity [Mbit/s] | 150 | **300** | 450 | 600 | 1000 |
| QoS{x} (AP) | 2 | **4** | 6 | 8 | 12 |
| QoS{x} (STA) | 0.5 | **1** | 1.5 | 2 | 3 |

standard settings to expose all performance-relevant functions, e.g. in terms of aggregation, in a static setup for architecture exploration. TXOPs are not necessary for low-bandwidth applications such as VoIP [202] or bi-directional traffic unless Reverse Direction Grant is supported [14]. Dynamic selection of aggregation strategies, e.g., based on packet arrival rates is subject to active research [138].

## 3.5 Model Verification and Characteristics

### 3.5.1 Measurements and Verification

Measurements most importantly include protocol conformance, i.e. real-time behavior, packet sequences, and frame formats must be verified. We have simulated and validated the model extensively in the defined scenarios. The framework allows detailed analysis and tracing of traffic and element execution. Applying assertion based techniques, a protocol checker monitors, e.g., response times for various protocol sequences. Data streams are checked for end-to-end packet loss, throughput, and latency. In addition, packet traces have been checked by tools such as Wireshark. Recently, the model has been successfully used for evaluating MAC-layer extensions for collision-free operation [1].

Exemplary simulation results further validate the model. The maximum delays for the QoS applications from Table 3.3 are met for all scenarios. Figure 3.5 shows the maximum delay incurred in the QoS scenario with two HDTV/VoIP streams and a packet drop rate of 5 % as the background traffic load (AC 0) is increased. While all higher QoS classes remain unaffected, the background latency is first large due to the A-MSDU aggregation timeout,

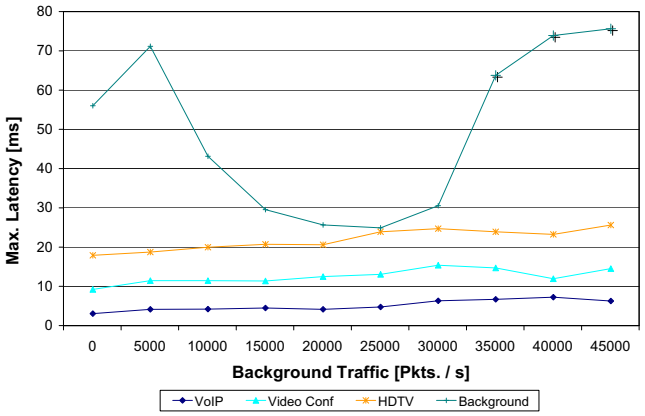



***Figure 3.5*** *– End-to-end latencies as background traffic is increased. For background traffic, timeouts cause large latencies at low loads and packet loss occurs at high loads.*

then comparable to AC 1 and lastly starts to grow as the channel capacity is reached. Packet loss only occurs for AC 0 as indicated with shaded markers. Finally, Table 3.6 shows that the number of collisions increases with the number of clients connected. The load on an ideal channel approaches theoretical bounds for the HT scenario.

***Table 3.6*** *– Simulation Performance.*

| Scenario | Simulation[1] (1min) | Load on air[2] | Collisions |
|---|---|---|---|
| HT (IMIX/AC 0) | 47 s | 253 Mbit/s | - |
| HT (IMIX/AC 3) | 23 s | 17 Mbit/s | - |
| QoS | 58 s | 185 Mbit/s | 145/s |
| Cmplx | 432 s | 38 Mbit/s | 404/s |

[1]) CRC and WEP disabled [2]) 300 Mbit/s channel

### 3.5.2 Model Complexity and Simulation Speed

All benchmark scenarios are generated from 18 Click source files with a total of 1188 lines of code (LoC). This includes verbose output and tracing. The library comprises 80 different elements with a total of 10722 LoC, see Table 3.7a. The number of instantiated elements in the AP varies with the number of STAs as shown in Table 3.7b. Most elements are inferred by hierarchy and macros. Despite the size of the models, the effort for customizing stations and modeling new scenarios is well within a few hours. Simulation speed depends on model complexity. One minute traffic takes between 23 s and 432 s on a 1.7 Ghz Pentium M system running Cygwin (see Tab. 3.6). This is sufficient to verify most scenarios in real time and even allows testing of large configurations.

## 3.6 Chapter Conclusion

A comprehensive and performance-indicative reference for the WLAN MAC layer and its surroundings was developed [6]. Such a reference was not available before, and it has

| *Library Type* | *Elements* | *Lines of C-Code (LoC)** |
|---|---|---|
| WLAN-specific | 41 | 7490 |
| Standard | 39 | 3232 |
| Sum | 80 | 10722 |

**(a)** Size of application library, (*) Physical LoC without comments and empty lines.

| *Scenario* | *AP* | *STAs* | *Env.* | *Total* |
|---|---|---|---|---|
| 1 HT | 118 | 123 | 44 | 285 |
| 2 QoS | 230 | 123 | 97 | 819 |
| 3 Complex | 1238 | 123 | 599 | 5741 |

**(b)** Functional element instances for scenarios of Section 3.4.

***Table 3.7*** *– Click application library and model characteristics.*

already been well received in the context of realistic evaluation of protocol extensions [1]. Realizing the model gave a better understanding of the WLAN protocol, capturing relevant aspects in an executable specification and exposing complexities. The model was verified in terms of protocol conformance and expected network performance. It will serve as a reference application for subsequent development steps. We highlight:

- The ***fully-functional IEEE 802.11n model*** covers all performance-relevant aspects for legacy operation, QoS, and high-throughput extensions. The WLAN library features 41 elements, e.g., for channel access, frame encapsulation, aggregation, and basic management functions. The modular concept exposes processing paths and – together with parameterization – allows for a range of configurations.
- The main ***complexities*** concern application timing relative to external events and MAC/PHY-layer aggregation including packet replay. These aspects are captured in a performance-indicative way. A feasible solution to (re-)configuration was proposed that keeps TID/QoS traffic streams explicit.
- ***Benchmark scenarios*** were defined that can be scaled to the device configurations of Section 2.3 in easy-to-use, static setups. These setups add detailed configuration, traffic environments, and the measurement facet to instances of the system function. The scenarios stress WLAN architectures with corner-cases related to scheduling (QoS), high-throughput (HT), and complex network topology.

The choice of **Click** as modeling language was essential for both productivity and applicability in our UMAC approach. Click allows to capture timing, inter-dependent streams, and complex state-machines in a modular and executable way. Our extensions regarding timing, modeling on a time-consuming resource, communication, and reconfiguration will facilitate deployment on resource-constraint multiprocessor systems. In addition, the setups comprising over 800 elements can be simulated in real-time, allowing to assess and verify long-term effects. Click is thus well suited for developing shared-medium MAC protocols, reinforcing previous findings from literature.

The total **development effort** for all performance-relevant features and the benchmark scenarios can be estimated to 8 effective man months (MM). This includes initial development (4 MM), integration and optimization of the aggregation mechanisms (2 MM), and identification and parametrization for the benchmark scenarios, e.g., adding flexibility to the model (2 MM). Considering recurring efforts for a new shared-medium protocol of comparable complexity (i.e., in the UMAC context), an improvement of 25 % for initial development is realistic thanks to the extended library and better modeling capabilities. If the new protocol has no complex mechanism such as aggregation, recurring efforts total 5 MM. Changes to the model setup (as, e.g., in Chap. 7), protocol extensions (as, e.g., in [1]), or developing a closely related protocol are a matter of days or few weeks.

# 4 Application Analysis

Having modeled the WLAN protocol function in the preceding chapter, step three of our methodology now derives architecture-independent application characteristics and requirements. We initially conducted a static performance analysis [10] that indicated the feasibility of software-based WLAN MACs. However, further evaluation of dynamic, architecture-dependent effects is necessary. This will be performed during the architecture exploration of Chapter 6, levering extended features of the SystemClick framework, e.g., for profiling and memory simulation.

In this chapter, more technical input is provided beforehand to guide exploration and implementation: The timing at the MAC/PHY interface is evaluated in the context of a given PHY architecture and critical processing paths in the WLAN application. This enables validation of design points and is essential for eventual deployment. Memory is a major factor for chip costs. Total requirements are thus determined in preparation of the exploration of their distribution and dynamic performance impact in Chapter 6. By analyzing concurrency, common functions, and memory access characteristics in the reference application, the architectural design space can be narrowed and necessary exploration steps can be identified. In addition, tool requirements become apparent.

## 4.1 Timing Requirements of Wireless Protocols

Timing requirements and critical processing paths must be reconsidered in the context of the MAC/PHY interface. Unfortunately, this interface is not standardized. Thus, we sketch the timing of an existing PHY and re-analyze transmission, reception, and channel access. From this rather technical analysis, performance and response time requirements needed for further exploration become available in Section 4.1.2 on Page 49.

### 4.1.1 Detailed MAC/PHY Interface Timing

The observable timing on the wireless channel is defined in the WLAN standard. Transmission times are determined by preambles, signal fields, transmission rate, and packet length (see also Sec. 2.1.5). Table 4.1 summarizes the channel timing for 802.11n packets. Critical are channel access and response times, which are discussed in the following.

Since the MAC interacts with the channel indirectly via the MAC/PHY interface, only an analysis of these interactions yields actual timing constraints for the MAC implementation. We sketch the timing of a the PHY [121] that is also used for the prototype implementation. It has been optimized to require data as late as possible and to provide data as early as possible. This leaves more time for the MAC, which is usually neglected in wireless designs. It is a special case of cross-layer optimization that does not improve network performance

***Table 4.1*** *– Packet lengths and transmission times of IEEE 802.11 frames.*

| *Type* | | *Length [B]* | *Time on Air [ µs ]* | |
|---|---|---|---|---|
| | | | *Legacy/54 Mbit/s* | *HT-Mixed/600 Mbit/s* |
| RTS | Ready to Send | 20 | 24 | 39.6* |
| CTS | Clear to Send | 14 | 24 | 39.6 |
| ACK | Acknowledgment of packets | 14 | 24 | 39.6 |
| BA | Block Acknowledgment | 32 | 28 | 39.6 |
| DATA | Data frame (0 – 2312 B payload) | 34 – 2346 | 28 – 368 | 39.6 – 68.4 |
| A-MPDU | 2 – 64 aggregated frames | 168 – 65535 | – | 39.6 – 910,8 |

*) The minimum time is given by the transmission of preamble, signal fields, and one symbol (3.6 µs).

but reduces performance requirements. In fact, many commercial WLAN systems fail to meet timing [26] and academic analyses are overly pessimistic [126, 78].

**Transmission** The interaction for transmitting a frame from a MAC-layer perspective is shown in Figure 4.1. A *Go-Signal* is sent as an initial trigger 2 µs ahead of the actual transmission to allow the RF system to power up. It contains preamble information that can be chosen statically, such as basic information on network setup and transmission power. The preamble synchronizes transmitter and receivers (16 µs). The signal field contains packet details – the *frame context* – which includes rate information, packet length, aggregation settings, and PHY specifics such as guard interval information. It must be provided 5 µs prior to transmission in order to be protected by a CRC. Then, more training fields may follow, before actual *packet data* is required, again with a 5 µs offset for encoding.

- *Result:* Detailed packet information is only needed approx. 14 µs after the Go-Signal. At least 18 µs are available for providing packet data.

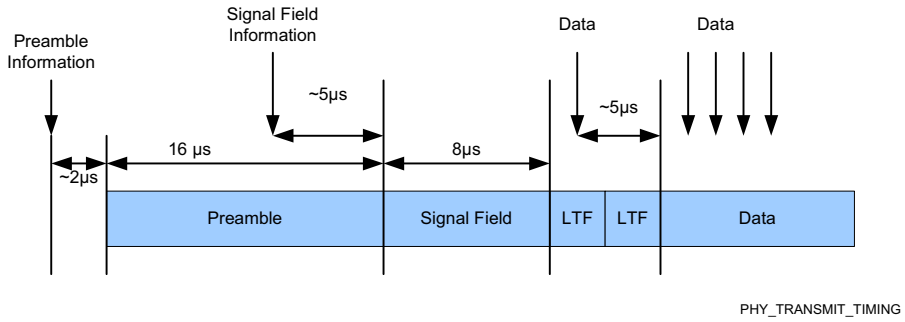



***Figure 4.1*** *– MAC/PHY timing for frame transmission in HT mode.*

**Reception** In receive mode, the PHY constantly monitors the channel for the start of a packet, i.e., its preamble. The *Clear Channel Assessment (CCA)* primitive indicates a *BUSY* or *IDLE* channel[1] and is sent to the MAC with an exact time stamp. The process is detailed in Figure 4.2. The signal field becomes available 3 µs after its reception. It contains the *RX Vector* with information about the received frame, which allows, e.g., prediction of the

[1] According to the IEEE 802.11-2007 standard [87], "the start of a valid 20 MHz HT signal at a receive level equal to or greater than the minimum modulation and coding rate sensitivity of -82 dBm shall cause the PHY to set CCA(BUSY) with a probability of at least 90 % within 4 µs".

frame end based on its length and rate. Packet end on air is indicated by CCA(IDLE). A recommended MCS rate is computed for rate selection. *Received data* decoding may induce up to 12 $\mu$s backlog

- *Result:* The frame end is known 12 $\mu$s before all data are available to the MAC, but processing can only start after the complete frame is verified.

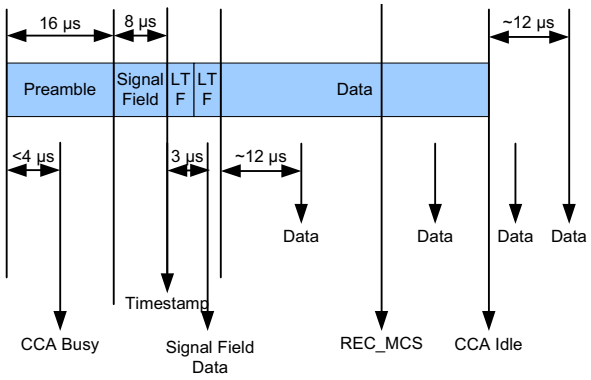



***Figure 4.2*** *– MAC/PHY timing during frame reception in HT mode.*

**Channel Access Timing** WLAN follows *Listen Before Talk (LBT)* for channel access (see also Sec. 2.1). Due to the MAC/PHY interface, the contention window access procedure from a MAC perspective becomes more complicated. During slotted operation, the MAC learns the CCA with a delay of 4 $\mu$s. When a packet is awaiting transmission and the back-off expires, it may be transmitted unless a busy channel (CCA) is indicated in the meantime. The Go-Signal must be asserted 2 $\mu$s prior to the packet's air time.

- *Result:* Only 3 $\mu$s remain for MAC processing during slotted operation unless special HW support, e.g., for pre-scheduling is available.

### 4.1.2 Resulting MAC-layer Application Timing

WLAN defines strict timing for frames on the air. Based on the MAC/PHY interface analysis, the actual budget for the MAC can be derived. Most relevant is the timing in two cases: 1) the gap between frames in sequence, and 2) the time for sending a response.

**Inter-frame gaps.** Sequences of frames sent out by one sender occur in different situations. *Same destination:* No gap occurs in between A-MPDU subframes. This is the worst case in terms of cycle budget per frame. The participants may agree on extra spacing to relax this requirement. All other sequences use SIFS between packets. *Different destinations:* Between frames destined to different stations within a TXOP, an interval of 2 $\mu$s (RIFS) is defined by the standard (SIFS may be used instead at slightly reduced throughput).

**MAC response time.** The SIFS time of 16 $\mu$s specifies the interval between a received frame and its response on the air. Considering interface delays, the data are available 12 $\mu$s after reception and a transmission must be triggered 2 $\mu$s ahead of time. This is summarized in detail in Figure 4.3 for our proposed PHY implementation. Consequently, a MAC implementation has the following time budgets:

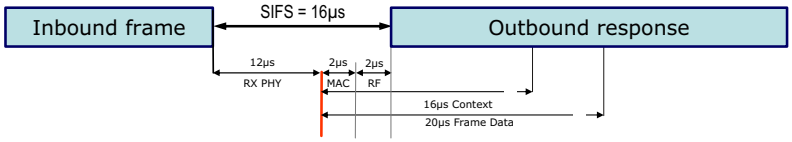



***Figure 4.3*** *– MAC time budget for response frames.*

- The *Go-Signal* specifying whether a transmission will follow must be sent after 2 $\mu$s.
- The *frame context* with information such as the frame length is due after 16 $\mu$s.
- The actual *frame data*, i.e., header and beginning of payload is required after 20 $\mu$s.

**Critical paths and budgets.** The resulting critical paths are summarized in Table 4.2. All channel reservation and legacy operations require a response time of 16 $\mu$s (SIFS). The budget for follow-up subframes in A-MPDUs depends on the size of the preceding subframe. In the unlikely case of minimum-sized Ethernet subframes (84 B), 1.1 $\mu$s are available at 600 Mbit/s PHY rate. The immediate generation of a BlockACK following a received A-MPDU aggregate is mandatory in 802.11n. Dynamic behavior that influences critical processing to meet the frame context deadline[2] is analyzed further in Chapter 6.

Rescheduling A-MPDUs immediately from received BlockACKs and dynamically scheduled multi-frame TXOPs are not mandatory, but essential for high throughput, efficient channel usage, and future systems (see Sec. 2.3). We thus deem them essential. However, if cost or performance concerns prevail, requirements can be relaxed seamlessly: 1) Restrict the length or the number of packet in A-MPDUs. 2) Do not consider a BlockACK immediately in the next aggregate. 3) Do not consider A-MPDU length dynamically or use static scheduling of single aggregates per TXOP.

***Table 4.2*** *– Critical processing paths in the WLAN reference application.*

| Description | Type | Mandatory | Gap | Comment |
|---|---|---|---|---|
| DATA->ACK | Response | yes | 16 $\mu$s | Legacy acknowledgment |
| RTS->CTS, CTS->DATA | Response | yes | 16 $\mu$s | Channel reservation |
| AMPDU-Subframe->Subframe | Follow-up | yes | >1.1 $\mu$s | Depends on packet sizes |
| (De)-AMPDU->BlockACK | Response | yes | 16 $\mu$s | Immediate BlockACK |
| BlockACK->AMPDU | Response | no* | 16 $\mu$s | Immediate response |
| AMPDU->AMPDU | Follow-up | no* | 2/16 $\mu$s | Only with TXOP continuation |
| AMPDU->CF-END | Follow-up | no* | 16 $\mu$s | Only with dynamic TXOP end |

*) but necessary for achieving maximum throughput

## 4.2 WLAN Application Requirements

The following section statically analyzes the reference application. In terms of performance requirements, an initial static analysis has been conducted [10] that indicated the feasibility of SW-based WLAN MACs. The complexity of the WLAN application and the dependency of the architecture on its behavior, however, necessitates a dynamic analysis of the full

[2] A closer analysis [10] showed that Go-Signal and channel access can be critical unless addressed properly: either a resource must always be available (cf. Sec. 6.4) or HW support is needed (cf. Sec. 6.6.2).

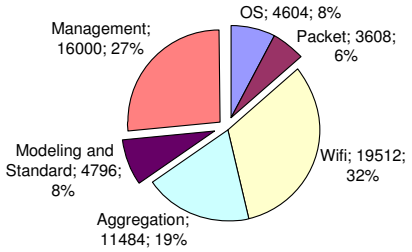



***Figure 4.4*** *– Code Size distribution of the 802.11n application (-Os switch, in [B]).*

system function. This will be based on a cycle-precise SW profile and further discussed in the exploration step of Chapter 6. Instead, we focus on memory requirements for code, data, and packet memory, as memories are a major contributor to chip area (and thus per-unit costs). Precise estimates are needed in preparation of the exploration step, where memory distribution and performance impacts are further analyzed and total chip areas are estimated for the reference WLAN configurations.

### 4.2.1 Code Memory Requirements

The code of the reference application is arranged into three segments: Start-up *Init* code, *Run*-time code, and *Shared* code that is always needed. Init code accounts for approx. 20 % of overall code size and can be swapped out after initialization. A small section (≈ 1 kB) for variable initialization is allocated to data memory. The **application footprints** are summarized in Table 4.3. The rising complexity becomes obvious: The 11abg setup is from an early implementation stage and the results have been published in [2]. The 11e setup is more basic in functionality but comprises already a more complex EDCA with QoS scheduling, resulting in a moderate increase in code size. The 11n setup is fully featured, effectively doubling code size over 11abg, with aggregation functions (A-MPDU/A-MSDU) being the biggest contributor. The operating system had to be extended to support advanced packet operations and, e.g., token communication.

***Table 4.3*** *– Code sizes (optimized for size, -Os) for different scenarios [in B]*.*

| *Scenario* | *11abg* | *11e* | *QoS(11n)* | *QoS(11n), agg. related* |
|---|---|---|---|---|
| *Run + Shared Segments* | 16358 B | 19856 B | 35792 B | 11484 B |
| *(Init Segment)* | – | – | 13080 B | – |
| *Operating System* | 6248 B | 8212 B | 8212 B | 1000 B |
| *Total Runtime Code Memory* | **22606 B** | **28068 B** | **44004 B** | **12484 B** |

*) excludes management functions

The **code distribution** in Figure 4.4 shows that WLAN-specific functions account for over 50 % of the overall size, with aggregation functions being a large contributor. This underpins the observation that the new HT aggregation functions are the main source for increased complexity in 11n. Another 27 % are used for WLAN management functions and interfacing. Standard Click functions account for 8 %. The OS including packet functions is below 10 %, which is reasonable given the tiny overall footprint.

Basic **management functions** (e.g., beacon generation) that are essential for stand-alone operation are comprised in the model and require around 12 kB (see also [2]). At the other

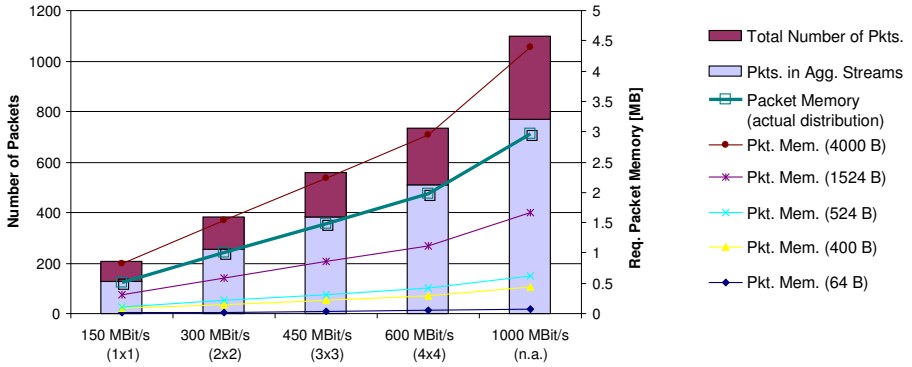



***Figure 4.5*** *– Number of packets present in an AP system for the home networking scenario QoS(11n) and the required packet memory for different packet sizes.*

extreme, a fully-featured driver of a 1x1 Lite-MAC [121] has 256 kB, yielding an upper bound for MAC-layer management functions. Higher layers such as *hostapd* [84] use several MBs of memory. However, non time-critical and low-performance functions are typically implemented on the host, which has dedicated external memory. This is even the case for Full-MACs, since the main concern is the performance impact rather than memory. We will thus assume a total of 12 kB as a placeholder for essential management functions or appropriate interfacing with higher layers in the AP (8 kB for STA).

## 4.2.2 Data Memory Requirements

### Traffic Environment and Packet Buffers

Packet buffering requires large amounts of memory. Understanding packets in terms of their number, size, and organization is thus paramount for efficient implementations. However, the number of packets present in a WLAN system is difficult to estimate. In literature, complex approaches such as [134] are common that consider, e.g., on network utilization, round trip times of TCP connections, and video stream buffering. Engineers use rule-of-thumb calculations and experience in practice. In our case, it suffices to consider the MAC part in a home network. Thus, the benchmark scenarios can be used, and the total number can be limited as most packets are buffered on the host.

For the estimation presented in Figure 4.5, the QoS(11n) setup from Sec. 3.4 is scaled with increasing bandwidth (cf. Tab. 3.5 on p. 44). It is assumed that 32 packets are required per active HDTV stream, 16 per video conferencing stream, 8 per VoIP connection, and 64 altogether for background traffic. A fixed amount of 128 mixed size (*IMIX*, [13]) packets is scaled with bandwidth relative to the 2x2 setup. This totals 384 packets for the 2x2 setup and almost 1100 packets for the future 1 Gbps setup. Approximately two thirds of the packets are queued in A-MPDU aggregation streams. Altogether, the memory required to store all packets is 1.0 MB and 3.0 MB respectively, assuming the actual packet distribution and A-MSDU aggregation rate derived from the QoS scenario. The requirements for a STA device can be lowered significantly, requiring only 144 packets and 250 kB of packet memory in a 2x2 setup.

***Table 4.4*** *– Memory for PDs (in kB depending on PD size) for single packets, packets with separate A-MSDU subframes, packets with A-MSDU subframes in segment lists.*

| *Scenario* | | | *Packets only* | | *Packets /w Subframes* | | ***Packets /w Subframes (64 B PD)*** | |
|---|---|---|---|---|---|---|---|---|
| *Throughput* | *Pkts.* | *Subfs.* | 32 B | 64 B | 32 B | 64 B | 64 B Lists | **32 B & 64 B Lists** |
| *150 Mbit/s* | *208* | *880* | 7 | 13 | 28 | 56 | 23 | 20 |
| ***300 Mbit/s*** | *384* | *1728* | 12 | 25 | 55 | 111 | 43 | **38** |
| *600 Mbit/s* | *736* | *3424* | 24 | 47 | 110 | 219 | 84 | 74 |
| *1000 Mbit/s* | *1100* | *5131* | 35 | 70 | 164 | 328 | 126 | 110 |

### Packet Descriptors and Organization

Packets are expensive to store and move within the system due to their size. They are thus represented by *packet descriptors (PDs)* that define the packets' data and that hold sideband information such as timestamps and annotations. The packets' data can be distinguished into the MAC protocol-specific *packet header* and the transported data, i.e., the *packet payload*. The PD format[3] used throughout this thesis requires 32 B without and 64 B if the packet header is included in the PD. Having the header available locally in the PD will prove beneficial during the exploration of Chapter 6.

The size and organization of PDs – especially regarding aggregated A-MSDU frames comprising many subframes – is a critical design decision, since the required memory is significant: ranging from 13–70 kB if aggregates are copied and represented with a single PD, but up to 328 kB if each subframe requires a separate, linked PD (Table 4.4, left and middle). This can be optimized as follows: The first payload segment is part of the PD and follow-up segments can be organized in separated lists linked to the main PD. This allows to concatenate and split packets without moving their payload, e.g., for A-MSDU aggregation (see Sec. 6.6.3 for a discussion of the performance impact). Fixed 64-byte lists suffice for close to all aggregated packets and are easy to manage. However, using also 32-byte lists adds flexibility to reduce memory. The total memory requirements for PDs can be reduced by 65 % if 64-byte PDs are used together with 32/64-byte segment lists. The 300 Mbit/s setup uses 38 kB in total. Other system designs may suffice with fewer memory, e.g., 24-byte PDs but typically lack a flexible approach to packet aggregation. As the total number of PDs is limited and as they have a fixed size, 2-byte pointers can be used for their addressing throughout the application.

### Queue and Management Memory

The model includes memory statically allocated to queue arrays[4] for every access category (128 entries) as well as for every TID (32 entries each) and every active BlockACK/TID stream (queues for retransmission, replay and reorder operation, 64 entries each). This ranges from 4.8 to 22.8 kB for AP configurations and compressed 2-byte pointers. Significantly less is

[3]The PD contains the packet length and a payload reference as a minimum. Sideband information includes annotations for Click communication, classification results, rate, and retry counters. Flags indicate aggregation status and, e.g., CRC and encryption information. A time stamp of 4 B is required for timing-exact processing. The packet data may comprise arbitrary combinations of *payload segments*. Assuming 8 kB as maximum size, the segments can be represented compactly using 6 B: a pointer to the actual location (3 B for addressing up to 16 MB), the starting offset (1 B suffices for header manipulations), the end offset (13 bit), and the segment size (3 bit to denote sizes as powers of two, starting from 64 B).

[4]Queues implemented as linked lists incur a memory and runtime overhead and are thus not considered.

needed for LiteMACs (1.6 to 4.4 kB), as queuing is done by the host, and for stations (3 kB), as less QoS streams are supported. Also, management data structures, e.g., for available rates and configurations of connected stations are needed – the StationInfoBase (SIB, cf. Sec. 3.3.6). The SIB can be kept small as configuration is decentralized in the Click elements. 128 B per station are assumed for short-time rate selection.

The 2x2 AP configuration needs a total of 19 kB for queues and management. An enterprise-level fully-featured AP may require up to 100 kB. For Lite-MACs and stations a fraction of the SIB suffices (only 3.1 kB for a minimal 1x1 Lite-MAC with 16 clients).

### Total Data Memory Requirements

An additional analysis based on the identification of distinct pieces of memory – *memory areas* – allows us now to summarize the total data memory requirements. These areas are available within SystemClick from profiling (cf. Sec. 5.4.4) and allow efficient exploration of memory mappings. The areas plus the previously identified requirements for initially/statically allocated memory are shown in Table 4.5. A small section is statically allocated to stack and data segments. Few helper structures are needed on the heap, such as template PDs and packets for control and management functions. The application configuration and state is stored in the element/port area (we state the optimized values from Sec. 6.3.3). Global/management memory supports up to 32 stations in the AP. Total functional memory also includes 38 kB for packet descriptors.[5]

***Table 4.5*** *– Total functional data memory requirements for the 2x2 reference Access Point and Station in the QoS(11n) scenario (in bytes), excluding packet storage.*

| *Memory Area* | *2x2 Access Point** | *2x2 Station*** | *Cost per agg. stream* |
|---|---|---|---|
| *Stack* | 1536 | 1536 | – |
| *Static data segment (ELF)* | 1024 | 512 | – |
| *Local heap* | 600 | 168 | – |
| *PDs and payload (static)* | 2592 | 1359 | 64 |
| *Static communication tokens* | 256 | 256 | – |
| *Click element structures and ports* | 14404 | 7076 | 572 |
| *Queues (2B pointers)* | 8448 | 3072 | 448 |
| *Global mgmt. and config. structures* | 10424 | 808 | 16 |
| *Total static memory* | **39284** | **14787** | 1100 |
| *Packet descriptors (dynamic)* | 37888 | 12544 | – |
| *Total functional memory* | 77172 | 27331 | – |

*) Includes 16 resp. **) 4 supported aggregation streams.

The total memory required for the 2x2 AP and station are 77 kB and 27 kB. The station is thus significantly smaller than the AP (only 35 %) and is dominated by its code size (44 kB), whereas the AP is dominated by functional data memory. It would thus be advisable to implement more of the station functions on the host to reduce code size (as often done in Lite-MAC designs). On the other hand, the focus should be put on data memory for an AP. For example, reducing modularity (the AP consists of 254 elements requiring an average of only 57 B for element state and ports) or by efficient organization of PDs as discussed previously. In addition, the memory statically allocated for one aggregation stream in our implementation is approximately 1100 B and includes queue space as well as elements

[5] Memory for packet payloads will be allocated outside the MAC due to its size. Similarly, beam-forming matrices, e.g., 50 kB for 4x4 MIMO are considered a functional part of the baseband processor (PHY).

(and state) for A-MPDU and A-MSDU aggregation. This can add up significantly for large configurations, underpinning the importance of reconfiguration as discussed in Section 3.3.6. Final results also for other configurations are discussed after exploring memory mappings and distributions in Chapter 6.

## 4.3 Application Characteristics

### 4.3.1 Application Structure and Implementation Characteristics

Exploiting concurrency increases throughput and minimizes response times by parallelization and better resource availability. It is therefore essential for efficient embedded designs, where tight constraints are put, e.g., on operating frequencies. Parallelism in the definition of Mihal [159] is mostly found at the task-level in the WLAN application.[6] From the application graph (cf. Fig. 3.3) receive and transmit parts can be distinguished. Due to the shared nature of the medium (half-duplex) and the resulting connections in between these parts, however, the elements related to medium access must run sequentially. This motivates the introduction of **two sublayers**[7]:

- The *Access and Response (AR) Layer* includes all elements and functions that directly participate in medium access and response generation. These elements need to run sequentially under timing constraints. Essentially, all elements after respectively before the queues and close to the medium in Fig. 3.3 belong to this layer.
- The *Preparation and Management (PM) Layer* includes all other elements closer to the host that perform preparation or post-processing (e.g., aggregation or encapsulation) and management (e.g., beacon generation). These are not subject to time constraints but impact the throughput of the system.

The two layers can run independently, and more parallelism is contained within each layer. Especially the preparation (and post-processing) functions are mainly data/packet-driven and can thus be parallelized or pipelined. At the AR layer, the scheduling of A-MPDUs can be decoupled in certain cases. This is further explored in Section 6.4.

Analysis of the **functions** in the model shows that WLAN is representative, complex, and generic. All core and extended functions (cf. Sec. 2.2.1) are comprised in our model, underpinning that WLAN is representative for related protocols. For example, the need for fast responses and to maintain a copy of transmitted packets is typical for ACK/nACK protocols. In fact, WLAN is rather complex and potentially a superset of other standards. For example, aggregation necessitating advanced packet operations and time-critical channel access proved time consuming during implementation. At the same time, the functions in their entirety are generic since diverse operations are needed including queuing, scheduling, bit field operations, packet modification and concatenation, timed state machines, classification, time computations (multiplications). Thus, no tendency towards a specialized or hard-wired target architecture can be identified before further exploration. Only payload moving and

[6] Instruction-level parallelism occurs, e.g., during lookups or loops, but exploitation is difficult with standard building blocks (keep it simple). Data-flow dominated payload processing such as crypto exhibits well-known instruction- and bit-level parallelism and will be considered accordingly during exploration.

[7] In [207], these layers are referred to as *Data Link Layer* and *Transaction Layer* respectively. This is reasonable in the context of IO interfaces but can be confused with the OSI layers in a network protocol context.

***Table 4.6*** *– Accesses of the WLAN reference application to shared data (management/configuration structures, queues, and packets).*

| *Layer* | *Element* | *Function* | *Access to Memory Area**** |
|---|---|---|---|
| *PM-Layer (TX/RX)* | EtherLookup | classify dest. station | global (SIB) |
| | WifiEncap/Decap | header translation | global (SIB, WirelessInfo) |
| | Rate Selection | observe link quality | global (Rates, Statistics) |
| | A-MSDU | check SIB, append packets | global (SIB), packet (Payload)* |
| | De-AMSDU | parse and split subframes | packet (Payload)* |
| | (Shared)Queue | enqueue / dequeue | queues |
| *PM-Layer (Mgmt.)* | Various | e.g., Beacon and Association | global (SIB, WirelessInfo, Rates), packet(Templates) |
| *AR-Layer (TX)* | ReplayBuffer | dequeue agg. queues | queues, global(SIB) |
| | A-MPDU | preview aggregates | queues, global(SIB) |
| | (Shared)Queue | check / dequeue | queues |
| *AR-Layer (RX)* | ReorderBuffer | reorder | queues |
| | EtherLookup | classify source station | global (SIB) |
| | BlockACK-Responder | format BlockACK | packet(Payload)** |
| | (Shared)Queue | enqueue | queues |
| *Payload Processing* | (De-)CRC | CRC32 calculation | packet (Header,Payload) |
| | Crypto | AES/rc4 | packet (Payload) |

*) depends on packet append/split function **) uncompressed BA only ***) excluding 32 B packet header

processing functions such as CRC and crypto are directly amendable to HW accelerators, as is often done in communication interfaces [207].

### 4.3.2 Memory Access Characteristics

The amount of required data memory (cf. Sec. 4.2.2) and the need to share data in a concurrent architecture motivates a closer analysis (as also discussed in [116]). Data shared amongst elements include packet descriptors and payload, global configuration, and (aggregation) queues. Leveraging SystemClick's memory area profiling (cf. Sec. 5.4.4), it can be shown that most protocol processing functions only require header modification (*header processing*), whereas CRC and crypto requires access to the packet's payload (*payload processing*). The initial 32 Byte of every packet as the *header* suffice for WLAN, leaving only the accesses summarized in Table 4.6 to shared data.

Since embedded (shared) memories trade off access latency with cost and are subject to arbitration effects, further exploration of these accesses is necessary (cf. Sec. 6.5). The throughput impact on the PM layer needs to be assessed and an appropriate data mapping must be identified. Especially A-MSDU aggregation necessitates advanced packet manipulation for split/merge. Since the memory accesses for split/merge depend on the choice of packet descriptors and packet organization, they are initially abstracted as a *platform service* (cf. Sec. 5.4.2). The AR layer must classify packets and is concerned with aggregation and access to queues that will impact real-time response times.

## 4.4 Chapter Conclusion

An architecture-independent analysis has been performed in preparation of subsequent exploration and implementation steps. This concerns especially **timing constraints** – which are difficult to derive since the MAC/PHY interface is not standardized – and **memory**

**requirements** – which is essential for determining the precise costs for different device configuration and architecture mappings:

- Abstracting the timing of an existing PHY, precise requirements for channel access and ***response constraints*** for application paths could be derived. They will allow validation of design points and help during prototype implementation.
- The ***code memory*** required for the 2x2 AP in 11n configuration is 44 kB, twice as much as for the 11abg setup. The biggest contributor are aggregation-related functions, indicating their complexity.
- The ***packet memory*** required is significant and on-chip buffering must thus be reduced to a minimum. Representing packets more efficiently, packet descriptors were introduced that support advanced packet operations.
- Functional ***data memory*** is estimated at 27 kB for a STA, but at 77 kB for an AP – dominating code memory for the latter. This indicates that for APs special consideration must be put on dimensioning and data structures such as queues and packet descriptors. For STAs, a Lite-MAC design can be favorable as it reduces code size requirements.

An analysis of the application graph structure, of common functions, and of memory accesses allows to derive a good **architectural starting point** and exposes **framework requirements** for a productive exploration:

- Two ***concurrent layers***, the Access-and-Response (AR) layer and the Preparation-and-Management (PM) layer, could be identified. The packet-driven PM layer can be pipelined as needed. Thus, a concurrent architecture and efficient ways to explore task assignment is needed.
- WLAN proves to be both complex and generic. Adding the need for flexibility (cf. Chapters 1 and 2) it is reasonable to start with a ***fully-programmable*** architecture template. This puts the (software) application into focus and thus necessitates a fast and precise performance feedback loop in the light of real-time constraints.
- The WLAN protocol can be divided into ***header processing*** and ***payload processing*** (CRC, crypto). These payload operations are candidates for HW accelerators and should be considered accordingly.
- Large data memories and shared state motivate an optimization of the ***memory hierarchy***. Dynamic access effects must be considered in order to assess the impact on throughput (PM layer) and timing requirements (AR layer).

In summary, a concurrent and fully-programmable architecture is to be explored that allows customization of the memory hierarchy and evaluation of HW/SW trade-offs. Such an architecture and an established tool flow will also be widely applicable in the universal MAC context, if it can be kept as protocol agnostic as possible. Analysis **efforts** for WLAN total 6 effective man months (MM). This includes a lengthy feasibility analysis based on profiling (4 MM) and initial architecture definition (2 MM). Recurring effort are estimated at 1 MM only for feasibility and memory analysis: better profiling tools are available (Chap. 5) and the prototype presented in this thesis has raised credibility for fully software-based solutions (Chap. 7). The MAC architecture defined in Chapter 6 can be reused.

# 5 SystemC-based Evaluation of Programmable Platforms

After modeling and analyzing the WLAN application in preceding sections, a suitable architecture template and an efficient framework is now developed to support the final steps of the application-driven approach – exploration and implementation. In fact, parts of this framework have already been successfully used for application modeling and simulation in Chapter 3 and initial performance verification and memory access analysis in Chapter 4. For the next steps it is again important to reuse application models in Click, which we already found best suited for complex protocol applications. Such a single model from specification to deployment will increase productivity, as it enables early exploration, facilitates verification, and will eventually provide a path to implementation.

The architecture-independent analysis motivates us to target a programmable multiprocessor platform for our MAC system, since such platforms are software-centric, flexible, allow to exploit concurrency, and encourage reuse. Thus, an existing network-optimized platform is chosen as the starting point. However, such a platform template must be customized by exploring architectural choices. A large design space is spanned, e.g., by multiprocessor mappings, memory distribution, and memory and communication subsystem characteristics. Especially memories pose a significant cost factor and their performance impact can even outweigh the choice of the processor.

Since detailed platform models (e.g., RTL) are both tedious to change for exploration and time-consuming to simulate, appropriate abstractions at the system level are needed. We leverage the *SystemClick* framework to abstract the performance of the target platform and to achieve fast turnaround of exploration steps. SystemClick allows fast changes to application-to-architecture mappings and quick evaluation of the full system function. It is extended as needed to meet the complexity, real-time, and productivity challenge posed by our application. This includes extensions to its modeling capabilities, a precise and fast performance loop, and efficient memory models. Finally, simulation performance and accuracy are improved and evaluated with respect to existing approaches.

## 5.1 Programmable Platforms

Programmable platforms, according to [68], are the next design discontinuity promising higher productivity, easier verification, and predictable results. They provide HW and SW templates that can be customized quickly, e.g., in terms of the number or type of processing cores or the memory hierarchy. Unlike HW/SW co-design [18], platforms orthogonalize concerns such that single aspects are easier to change and design decisions can be made later [107]. Key features are thus modularity, simplicity, and scalability. Surveys on packet processors and platforms can be found, e.g., in [242].

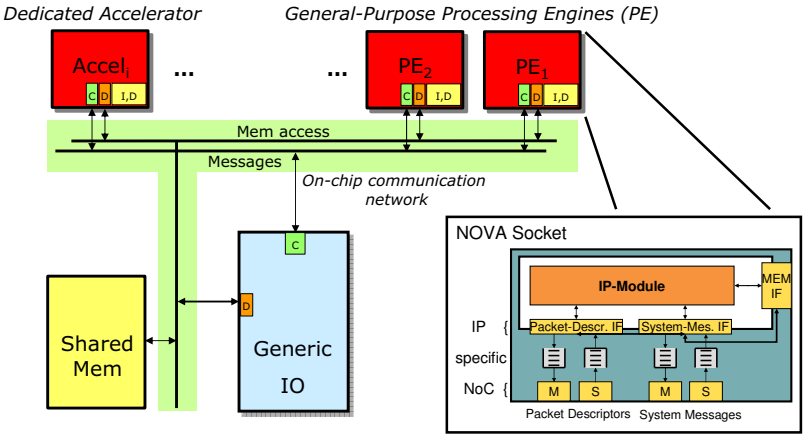



***Figure 5.1*** *– The generic NOVA platform template [208] with processor cores embedded in Processing Elements (PEs). The NOVA Socket decouples PEs and IP cores from the communication and memory subsystems.*

Few platforms emphasize programmability and modularity rather than high performance (as, e.g., [173, 171]) or targeting specific architectures (as, e.g., [11]). Examples include *Tensilica's Extensa*, which allows core customization, and the *StepNP* platform [183], which supports Click as a programming model. We favor the NOVA platform, since it is readily available and yields efficient packet-processing systems based on Click [207].

### 5.1.1 The Network-Optimized Versatile Architecture (NOVA)

For exploration and implementation of the WLAN/UMAC architecture in Chapter 6 and Chapter 7 we will extend on the NOVA multi-processor hard- and software platform [208]. Figure 5.1 shows its architecture template. We assume a variable number of general-purpose *Processing Elements (PEs)* for protocol functions. A shared memory stores packets and shared data structures. Further specialized nodes (i.e., co-processors or IO) may be added as needed. A central concept of NOVA is the *NOVA Socket*. It decouples IP cores such as processors and hardware accelerators from the communication and memory subsystems, facilitating the exploration of each of these facets.

#### Hardware Building Blocks

**On-chip communication.** Memory accesses and inter-element communication are kept separate by the communication network. For the latter, HW-supported message passing can be used. Following the Click communication semantics, messages are primarily used to transfer *Packet Descriptors (PDs)* of size 64B representing a packet between processing nodes. In addition, *System Messages (SMGs)* may be exchanged, e.g., for OS like functions.

**General-purpose PE.** A PE comprises a 32-bit processing core with local instruction and data memories (Harvard). Its register set includes hardware counter and timer registers. In addition, a memory interface (pipelined, split-transaction), a tiny DMA function, and the *Message Passing Interface (MPI)* with a set of hardware transfer queues are integrated

memory-mapped into the module. The DMA may be used to transfer messages from the MPI into the local memory and vice versa.

**Dedicated Accelerator.** Accelerators may be integrated into the platform for computationally complex or time-critical functions. These typically feature message passing interfaces and may have access to shared memory.

**NOVA IO Interfaces.** NOVA provides a generic template for IO interfaces that includes autonomous packet transport through DMAs and parsing of PDs. Extending on this, a *Specialized PE (SPE)* for WLAN will be developed in Chapter 6.

#### Programming Environment

NOVA supports Click as a programming model and provides code generation for embedded cores based on an element library written in C [210]. Specialized library elements, APIs, and extended code generators encapsulate HW specifics. A thin OS layer provides a runtime environment that we abstract as *platform services* as discussed in Section 5.4.2. Special purpose hardware, such as timers and co-processors, is encapsulated in a hardware-independent API. Depending on the way of coupling, hardware can be deployed either directly using the API within elements or indirectly by particular elements that are mapped onto the special purpose hardware (e.g., encryption). Apart from Click, OS-like services for system messages, task handling, exact timing, and resource sharing among several processing elements are needed. A direct consequence of the message passing paradigm is, for instance, the lean event handler and task scheduler, which exploits the Click run-to-completion and cooperative scheduling semantics. Events, such as arriving PDs, SMGs, or expired timers, trigger the execution of associated tasks that run until completed. The register file does not need to be saved.

### 5.1.2 Click on Concurrent Systems

Multiprocessor platforms such as NOVA are concurrent and can thus minimize response times and increase throughput by parallelization and better resource availability. This section discusses implications and the state-of-the-art of mapping Click onto such platforms. Click readily satisfies the requirements for a **programming model** [16]:

- *Identification* of computational and parallel tasks, communication in between tasks, and relevant data structures.
- *Mapping* of computational tasks to processing elements, distribution of data to memory elements, and communication to the inter-connection network.
- *Synchronization* and scheduling of tasks.

Click is originally single threaded [113]. Its *Model of Computation (MoC)* following push/pull semantics translates to function calls. Queues are explicit and connect pull to push chains that are scheduled by timers or IO. As stated by Lee [122], Click deals with concurrency such as converging asynchronous streams in a robust and natural way. Click's modular elements are combined to execution 'islands' [252], i.e., its modularity eases reuse but does not affect semantics. Thus, **Click's semantics** must be revisited for concurrent execution:

- Click's model of computation in terms of execution *at the graph or process level* is only defined properly for single-core execution:

  - Element connections mapped to function calls imply a strict processing order despite the modular approach and unlike *Process Networks (PNs)* [99, 123].
  - Timer scheduling and IO are based on *First-Come-First-Serve (FCFS)* and undefined if multiple requests occur concurrently or distributedly.
- Click is very expressive in that it does not impose restrictions on the function *inside elements*. Especially, packet input and output may be arbitrary, which is fundamentally different from actors [12, 57].
  - Every push/pull is an implicit synchronization point, i.e., the element yields control. Elements can thus be re-entered if the graph contains cycles.
  - Return paths of push chains are an implicit synchronization informing upstream elements that execution is finished on all downstream elements.

The consequence are two **well-known approaches** that either restrict Click's expressiveness or application complexity in order to avoid ambiguous cases. If simplified to actor semantics, elements from a library can be mapped directly to FPGAs [116, 201], where a communication and scheduling infrastructure is provided. A related approach re-implements elements in an actor language to exploit element and graph-level parallelism on sub-RISC processors [158]. Focusing on IP routers, Chen [39] maps Click onto a *Symmetric Multiprocessor (SMP)*. Expensive elements are duplicated or pipelined, which is only possible for linear and independent flows. Data structures in critical elements must be protected by locks. However, a different approach is needed for applications with dependent flows, complex state machines, and exact timing such as WLAN.

Together with Sauer [210] we **propose to adapt the application** itself to maintain the original Click semantics, e.g., for NOVA. Explicit insertion of queues to cut task chains forces the designer to consider implications such as reentrancy while allowing distribution of tasks on concurrent resources. Pull chains are either triggered on stock ahead of time or stall until a response is received, depending on the application. Communication must be reliable, e.g., relying on back-pressure if mapped to the target platform. This effectively bounds intermittent queues but can lead to deadlocks. If all concerns are properly reflected, we deem the approach well suited for WLAN.

For reduced response times, we introduce an extension to Click with priorities that exploits the fact that elements yield control at their boundaries. This *Click Element Threading (CET)* re-evaluates priorities only at such points. This simplifies task switches and allows critical tasks to be executed preemptively. CET is further discussed in Section 6.4.2.

## 5.2 Platform Evaluation, Exploration, and Implementation

### 5.2.1 Requirements

Exploration of architectural choices, e.g., by platform customization is essential for efficient designs. However, cycle-precise and RTL models of the target platform are lacking abstraction to change and evaluate platform instances productively. Thus, a more efficient approach to early exploration and performance evaluation of programmable platforms is needed. We recognize the following **basic requirements** of Sauer [207]:

- *Rapid generation of platform instance for exploration* – requiring separation of application and platform, support of Click, and evaluation of the full platform.
- *Early system-level performance evaluation* – requiring feedback from architecture timing to system function and a high level of abstraction.
- *Productive exploration and programming environment* – requiring a path to efficient implementations in hard- and software and an appropriate programming model.

Special challenges are posed by WLAN with respect to application complexity, real-time requirements, and productivity. We emphasize the need for supporting *Click*, which proved well suited for application modeling, to promote model reuse for increased productivity. It is imperative to evaluate the full system function, because protocol behavior can be influenced by the timing of the target platform (unlike homogeneous benchmarks such as JPEG). In addition, we add the following **refined criteria**:

- *Efficient performance feedback* – Due to the complexity of WLAN, optimizations regarding the system and software functions are much more common during exploration, necessitating a fast turnaround for profiling, performance simulation, and analysis at appropriate abstractions.
- *Memory and communication facets* – The WLAN application maintains complex state, manipulates packets, and relies on shared data structures. Thus, the memory hierarchy must be explored for efficient solutions and – together with communication – be reflected in the verification of real-time constraints.
- *Fast/accurate simulation* – On the one hand, real-time requirements must be evaluated precisely, on the other hand, verifying complex WLAN transactions necessitates long simulation runs. Thus, high simulation performance and trade-offs between accuracy and speed are needed.
- *Productive modeling* – Graphical design entry and graph transformations increase development efficiency in the face of a complex application. In a commercial environment, the use of SystemC [71] enables step-wise refinement and reuse, mixed-level simulations, and virtual prototyping.

### 5.2.2 System Development at the MAC Layer

Protocol design is done with performance-abstracting frameworks, e.g., *OmNET++* [235]. But exploration and development of protocol-processing systems and their architecture is often based on ad-hoc methods (see Sec. 2.2.3). We perceive related work in the fields of development frameworks, evaluation of HW/SW trade-offs, and the use of SystemC.

Many frameworks map protocol applications onto **fixed architectures**. They leverage high-level application descriptions and optimize performance but are missing the exploration facet. For example, *Shangri-La* [41] is a comprehensive compiler-assisted framework based on the language *Baker* targeting *Intel IXP* network processors. Similarly, *NPClick* [217] maps Click applications to the IXP. Click is also used for integrated development targeting an SDR platform in [47]. SDL-based approaches rely on a runtime environment, e.g., in [179] for WLAN. A specific operating system is often targeted for small-scale nodes of wireless sensor networks [133, 32]. Mapping descriptions of packet functions directly in hardware as in [224] is difficult for complex MAC applications.

Few authors perform **design exploration** at the MAC layer as discussed in Section 2.2.3. However, their approaches do not support Click or SystemC, lack precise performance feedback, or are inefficient. For example, the OCAPI-xl library is used for HW/SW exploration with abstracted performance for the HYPERLAN/2 protocol [152]. SDL models of an UWB system are manually encapsulated in SystemC and coarsely instrumented in [77]. SDL is also used in [48] for HW/SW partitioning, but the approach based on co-simulation is too slow for exploring programmable platforms. UML is used in [102] to evaluate a WLAN system, but the implementation targeting an RTOS is inefficient [15].

The use of **SystemC** is often limited, e.g., to model the network environment or to evaluate power aspects. Bombieri [31] and Damm [45] couple co-simulated TLM models of network nodes with network simulation. TLM models are also used in [21] to model a shared medium connecting embedded systems. Still, network simulation with SystemC is promising as it facilitates integration of detailed node models and verification.

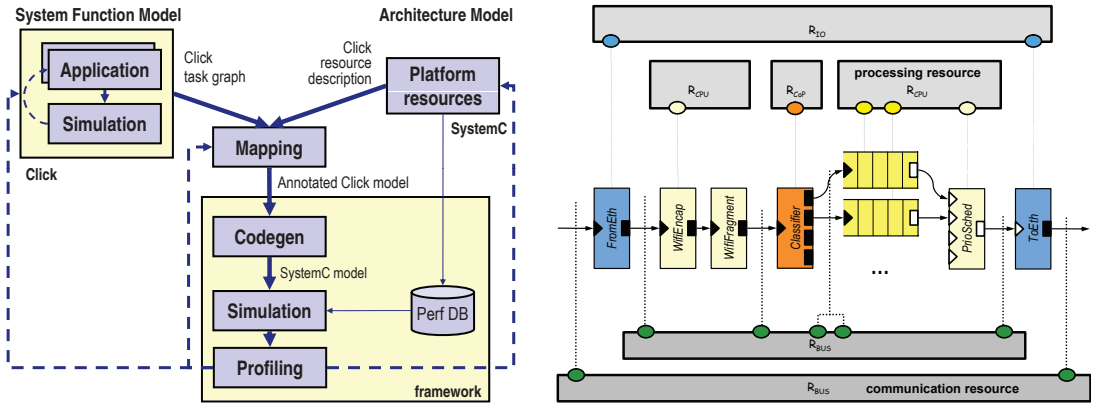
### 5.2.3 Existing Platform Evaluation and Exploration Frameworks

Outside the networking community, a plethora of comprehensive methodologies and tools is available. Many of these are generic or target signal processing domains. Approaches to **performance evaluation and simulation** alone are insufficient, since the exploration facet and an efficient path to deployment are missing. We give a brief overview here, but discuss details related to performance simulation and memory exploration relative to our work in Sections 5.4.6 and 5.5.5 respectively:

- *Abstract or static performance evaluation* – e.g., based on queuing networks [223], statistics [65], formal methods [213], graphs [119, 176], traces, or static analysis [145] do not consider the full system function and dynamic effects.
- *Co-simulation* – of the complete system function based on *Instruction Set Simulation (ISS)* as, e.g., in [63] is typically slow and does not allow early and fast exploration. The combination with TLM abstractions as presented in [211, 33, 11] still relies on fine-granular synchronization of memory accesses, and models are cumbersome to change for fast exploration.
- *Host-code execution* – speeds up processor simulation by executing the system function natively and back-annotating performance. It is thus the means of choice for fast exploration. Existing approaches, however, either abstract performance and do not provide accurate feedback [28, 182, 212, 186] or are too low-level and tightly synchronized for early and fast exploration, e.g., [64, 214, 106, 42]. Abstraction of memory accesses is only considered in [182].

A large number of **exploration frameworks** exists (see, e.g., [46] for an overview). According to Sauer[207], however, only frameworks based on the *Y-Chart* [109] leverage distinct descriptions for application, architecture, and mapping and separate concerns of application developers and platform architects. Given appropriate abstractions, designs can thus be explored and implemented quickly while facilitating reuse. Unlike top-down approaches found in HW/SW co-design [18] and refinement methodologies [243, 52], such a meet-in-the-middle approach can efficiently address the implementation gap between concurrent application descriptions and heterogeneous hardware platforms.

None of the frameworks reviewed by Sauer – including the comprehensive *Metropolis* [19] and the Click-supporting *Mescal* [158] methodology – were able to fulfill the requirements of




**(a)** Platform exploration with SystemClick.

**(b)** Mapping a Click model onto concurrent resources.

***Figure 5.2*** *– (a) Overview of the SystemClick exploration flow. (b) Following the Y-Chart, application and platform are kept separated and only brought together by explicit mapping.*

Section 5.2.1. Two more recent Y-Chart frameworks are promising, since they use SystemC and offer performance feedback. *SysCOLA* [239], however, only supports the automotive domain-specific language COLA, and the approach in [226] targeting signal processing is limited to actor-based languages. In addition, they do not support efficient exploration of the memory hierarchy, and the accuracy of [226] is insufficient for real-time purposes. Only *SystemClick* as described in the next section matches all basic requirements. It will be extended to cater our refined needs as proposed in Section 5.2.5.

### 5.2.4 SystemClick

We have developed SystemClick together with Sauer [9] to have independent descriptions of the system function and the platform architecture, such that different mappings from functions to hardware building blocks can be explored seamlessly. SystemC performance models (including function behavior) are generated from high-level descriptions employing library elements for application and hardware resources following the Y-chart, as sketched in Figure 5.2a. A performance analysis step provides feedback for optimizing architecture, application, and mapping. SystemClick is lightweight and easy to extend. The different views of the framework are explained next.

**System function in Click.** The application is specified in an architecture-independent way using Click. Applications are composed from modular elements linked by directed connections, as suited for describing packet flow and processing (cf. Chap. 3).

**Architecture models in SystemC.** Architecture blocks are specified as performance models that represent shared computation and communication resources. Each block is described by a *Resource Manager (RM)* in SystemC that arbitrates tasks and consumes time. Resource managers also specify the properties of a resource, such as its type, operating frequency, and scheduling policy. Abstract RM elements can also be inferred automatically in order to speed up the exploration process.

**Mapping and code generation.** Function and architecture specifications are used as input to the mapping step. The application graph is partitioned at element boundaries, and

task chains and communication are mapped onto an SoC platform manually (cf. Fig. 5.2b). Additional wrappers represent the operating system aspects of a platform. They model the overhead required for event handling and task scheduling. From the annotated source, the framework generates a SystemC model that is compiled together with libraries for application and architecture using standard GNU tools. Using a performance database, behavior and timing of the full system model can be simulated.

SystemClick covers modeling, exploration, and deployment from a single source of code. The flow is centered around the cross-platform code generation framework *Click Rapidly Adapted to C Code (CRACC)* [210], which extends the Click framework and is built on the GNU tools. Model entry and exploration are performed with Click models that are targeted to SystemClick's SystemC resources for simulation. The elements are implemented in CRACC, i.e., in C rather than C++ (which is used in the Click framework). Such an implementation is more indicative of performance regarding resource-constraint systems and can be targeted directly to embedded cores, leveraging the OS layer of NOVA.

### 5.2.5 Open Issues and Next Steps

Assessing MAC development approaches, we found that a more systematic and efficient approach is needed. SystemClick is best suited to our needs, as it is easy to use and to extend and allows for efficient evaluation, exploration, and deployment of programmable platforms such as NOVA. It supports Click as needed for describing the WLAN application and is based on SystemC.

However, a number of shortcomings must be addressed in order to make SystemClick applicable to complex real-time applications in a productive way. For example, resource managers have been improved by adding auxiliary interfaces and better client management. More schedulers, e.g., based on priority, and wrappers supporting FIFO interfaces and different flavors of pull connections have been introduced. Also, the performance database has been extended to support multiple configurations. From the requirements of Section 5.2.1 the following **main issues** remain and will be discussed in the following:

- *Modeling and mapping* – Graphical design entry is needed in addition to advanced mapping capabilities and modeling extensions to abstract on-chip communication. This will increase productivity and ease of use, and is essential for comprehensive evaluations. (Sec. 5.3)

- *Performance exploration with automated feedback* – Automated yet flexible methods are needed to characterize, simulate, and analyze Click applications in terms of performance and memory accesses while abstracting platform functions. This counters the complexity and real-time challenges posed by the WLAN application. (Sec. 5.4)

- *Memory models* – Memory is a primary cost and performance factor in deeply embedded systems. Efficient means for both configuration of the memory hierarchy and consideration of performance-limiting effects must thus be available. (Sec. 5.5)

- *Simulation performance* – WLAN has real-time constraints but its complexity necessitates long simulations. The framework's performance must thus be increased and the accuracy must be re-assessed in the light of newly added extensions. (Sec. 5.6)

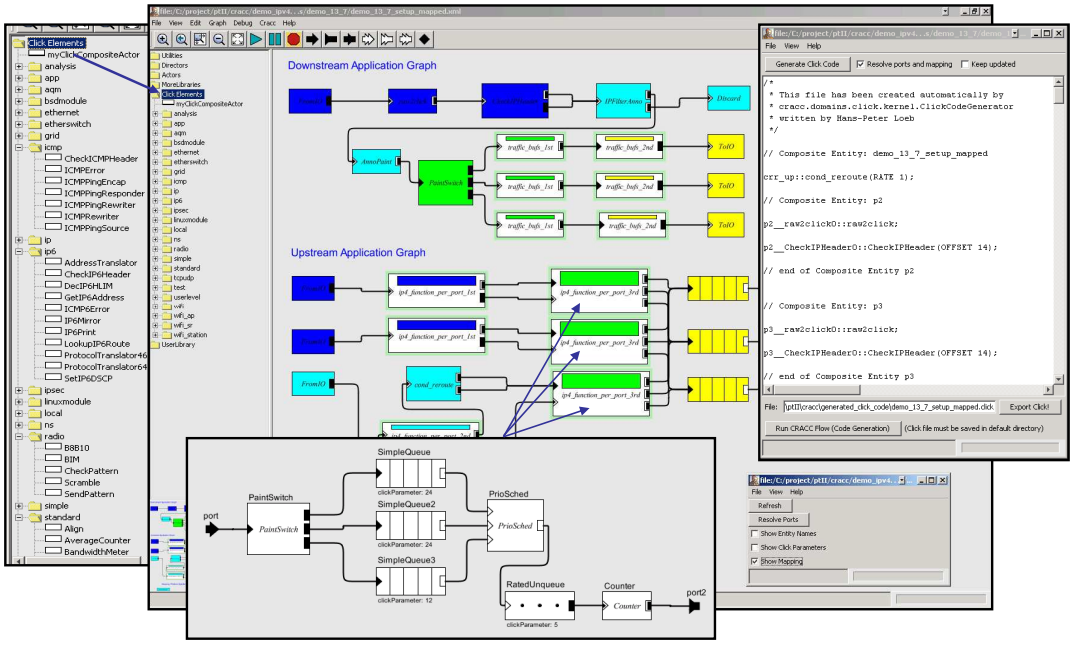



***Figure 5.3*** *– CliMMT with element library, hierarchical Click model, and its graphical and textual representation. The mapping to platform resources is indicated by colors.*

## 5.3 Modeling and Mapping

In preparation of the major extension steps – performance feedback and memory modeling – we present our graphical tool *CliMMT* and discuss extensions to SystemClick that increase expressiveness, including modeling capabilities for system communication.

### 5.3.1 Graphical Modeling of Click Applications

Design entry for Click models and their mapping so far is a textual process. But Click has a natural graphical syntax, i.e., elements and connections that can be exploited for productive modeling. Few dedicated tools for Click exist such as *ClickController* and *Click!It*, but they are immature and require a live Click configuration. Thus, we have developed a **graphical, component-based editor** – the *Click Modeling and Mapping Tool (CliMMT)* [8] – and integrated it into the SystemClick framework.

Graphical editors are tedious to implement from scratch. But generic modeling frameworks such as *Eclipse/GMF* or [55] are overly complex, require metamodels, and do not support hierarchical modeling. Instead, CliMMT is based on Ptolemy [57], because it is easy to extend, provides a graphical infrastructure, supports hierarchical modeling, and represents the model in XML. Existing actor domains such as [252] could not be reused as they lack, e.g., proper graphical representation.

CliMMT has its own representation of Click elements, extends on the idea of a component library, and introduces capabilities for validating, transforming, and mapping Click application graphs. Figure 5.3 shows the graphical Click model, the extended element library, a

hierarchically zoomed compound element, extended options, and generated textual Click code. Summarizing **CliMMT's features**:

- Easy-to-use *hierarchical modeling* of Click graphs with drag-and-drop, model overviews, zooming, and saving/loading of configurations. Configuration strings can be configured and are propagated into compound elements.
- Large *template library* of basic elements parsed from the Click library (including port and configuration descriptions) and user-defined compound elements.
- *Object orientation* extending modularity and abstraction to modeling with classes, inheritance, interfaces, and polymorphism [125]. Changes to classes and predefined configuration strings are propagated to instances.
- *Graph checking* for correctness and resolution of agnostic ports.
- Textual Click *code generation* from graphical syntax and parsing of existing models.
- *Mapping* by assignment of colors representing platform resources. Mappings are propagated in downstream direction until a manually mapped element is found to avoid re-mapping. Platform-specifics are encapsulated by placeholder elements.
- *Graph transformation*, e.g., insertion of push-to-pull converters at cuts or insertion of additional elements with information for advanced mapping.
- *Tool integration* includes seamless code generation for SystemClick simulation, embedded cores, and the NOVA platform.

The **mapping step** has been extended over [210]. Graphical input is supported as explained above. Element connections can be mapped to communication resources by inserting mappable *cut* elements, e.g., representing FIFOs (see Sec. 5.3.2). Data structures are represented by special elements that can be assigned to storage resources (see Sec. 5.5.2). In addition, mapping of arbitration parameters such as priorities is introduced (needed for CET in Sec. 6.4.2). These are propagated downstream and rely on Click's *StopFilter* to detect already mapped elements. For priorities, if a given element has inputs with differing parameters, the maximum priority must be assigned. The graph can be automatically cut at every element boundary. After flattening, appropriate wrappers are inserted, or placeholders with information, e.g., on the nature of a connection (push/pull) and its priority are dumped so that they can be processed by scripts.

### 5.3.2 Extensions for Communication Modeling

SystemClick is lacking features to model application communication, as is vital for indicative platform evaluation. Click models are partitioned onto concurrent architectures by cutting task chains (cf. Sec. 5.1.2). Cut tasks communicate packet contexts. Thus, communication must be mapped onto SoC interconnect. In our framework, wrappers extend task chains at these cuts and pass packets to communication infrastructure modeled in SystemC. Also, wrappers trigger execution as packets become available, indicated by events. We provide such wrappers for transactions of packets over standard signal and FIFO interfaces that can be refined if needed. These may be used explicitly by the designer to partition, refine, and map the communication, or be inferred automatically.

A natural match for the communication semantics of Click's model of computation is a reliable message passing architecture with back-pressure. In Figure 5.4, a cut task chain is

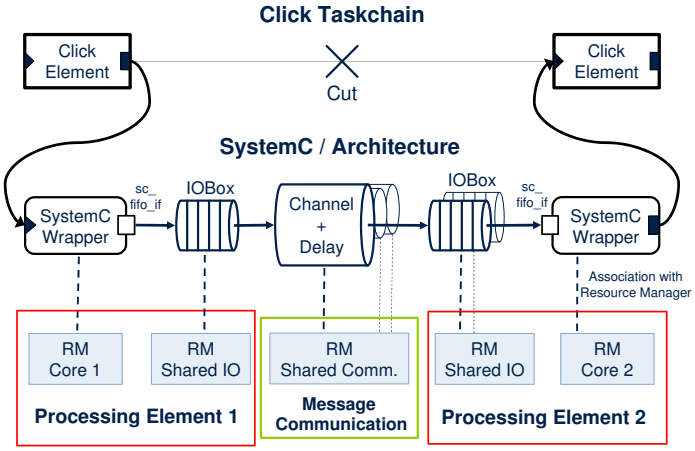



***Figure 5.4*** *– Mapping Click element connections to a shared communication architecture.*

mapped onto a **virtual communication system** replacing the original connection. Default wrappers are connected with *I/O boxes* that extend SystemC's FIFO channels. I/O boxes may arbitrate ports. If associated with the same resource manager, they can share their capacity. In combination with FCFS arbitration the behavior of a PE's shared in- and outboxes is modeled. To reflect a simple shared medium connecting the I/O boxes, we use a hierarchical channel that associates with an RM to arbitrate its transport function. This channel also progresses simulation time by a transport delay that may depend on transaction size and the RM's operating frequency. Optionally, connecting channels can be assembled from basic SystemC modules or fully refined, e.g., to an NoC.

Our approach preserves the direct connections between task chains, which avoids system-wide addressing and improves performance but necessitates **extensions to SystemClick**. A communication may only take place if both the medium and the target inbox are available. ResourceManagers (RMs) must thus offer non-blocking locks and a semaphore-like capacity interface that notifies if capacity becomes available again. In addition, *concurrent RMs* have been introduced for fairness if locks occur concurrently in tightly synchronized parts of the model (see also clocked RMs in Sec. 5.5.3). These RMs defer arbitration to the next SystemC update phase after no more requests are registered. The last update of a delta cycle, as would be needed ideally, is not accessible in SystemC [71]. Similar to our work, the lack of non-consuming reads and a guard function in *sc_fifo* is also identified as a problem for modeling communication in SystemC by Wang [239].

### 5.3.3 Discussion

The CliMMT tool proved very useful during development of the WLAN application. In addition, it increases the acceptance at potential users that are put back by textual representations. Dependencies, similarities, and hierarchy of the application become apparent and reuse is thus encouraged. During development and deployment, changes to the mapping or the application structure can be done within minutes. With growing complexity, however, graphical syntax can become tedious to use. Thus, a bi-directional automated way between graphical and textual syntax is essential. CliMMT is extensible and can be used to visualize

performance or memory use of elements (e.g., as in [171]). Efforts exist to visualize the flow of packets in Click graphs [241].

The advanced mapping mechanisms proved essential during exploration. Different cut variants opened more options for parallelization, and priorities enable better processor utilization for low-cost solutions. The communication mapping can have a significant impact on real-time performance if not considered properly (Sec. 6.7).

## 5.4 Performance Exploration with Automated Feedback

With increasing complexity the application itself becomes an essential aspect of the exploration process. Appropriate abstractions for software, fast performance simulation, and a precise performance feedback loop are thus vital for productive platform exploration. SystemClick is back-annotating performance and memory accesses during simulation to reflect the target platform's behavior. However, a quick and systematic way to profile applications and to analyze their behavior is missing. Targeting early exploration, our approach must be flexible and offer trade-offs, as part of the system function may not yet be fully available and promising exploration steps must be identified in the first place.

### 5.4.1 Performance Simulation Feedback Loop

Our extended flow integrates all essential aspects, namely profiling, simulation, and analysis into a **performance simulation feedback loop**, as outlined in Figure 5.5: It is based on a single source of code (1) with instrumentation points (see Sec. 5.4.3) for automated profiling and simulation. During initial profiling (2), parameters such as selection of worst-case or average performance and the profiling method can be given to determine computational and memory access profiles. These are stored in a database for subsequent simulation runs. Platform functions, e.g., for packet manipulation must be abstracted and made explicit, since the final implementation is typically not yet available. Assumptions on their performance or on future optimizations can also be entered in the database. During subsequent simulation (3), the profiled code is executed while performance is back-annotated from the database at instrumentation points. For analysis, both statistical data from the profile or dynamic simulation traces are available (2a and 3a). Since all aspects are automated, optimizations to the source code (4) can be quickly re-evaluated, either by changing assumptions in the existing database or by re-profiling. All aspects are discussed in the following, focusing on the following **essential features**:

- Precisely *capture* and back-annotate computational and memory access *requirements* at code instrumentation points – at the granularity of relevant blocks and relevant data structures. Accurately consider target platform specifics (e.g., compiler, processor) while abstracting platform features that are not yet available.
- Allow *trade-offs* between automation and flexibility, i.e., inclusions of assumptions and additional instrumentation at points of interest, and between accuracy and speed, i.e., instrumentation granularity and different profiling approaches.
- Provide simulation-based and static *analysis* methods at appropriate abstractions to help understanding the application, identifying exploration steps, and debugging.

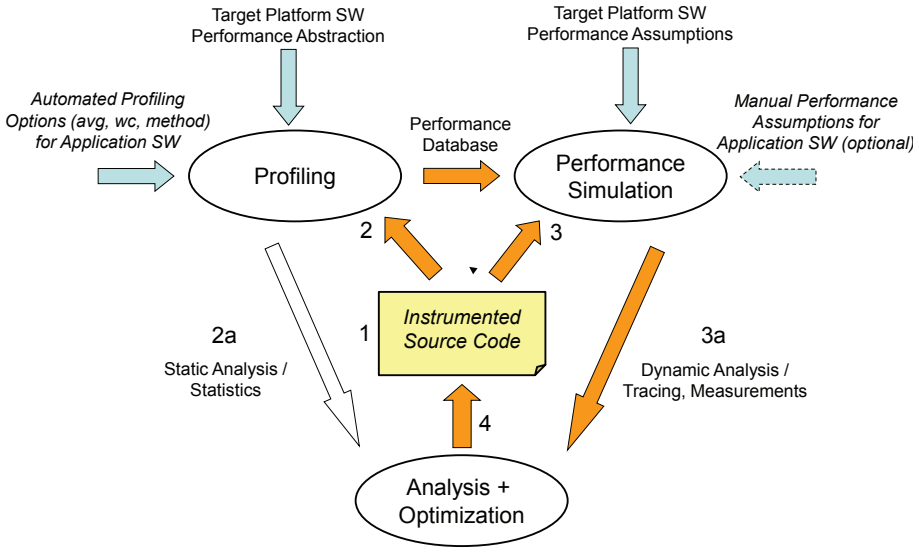



***Figure 5.5*** *– Performance simulation feedback loop relying on instrumented source code (1) for both automated profiling (2) and subsequent simulations (3). Performance assumptions can be added manually to generated performance profiles stored in a database. Both static (2a) and dynamic means (3a) can be used for analysis and optimization (4).*

### 5.4.2 Platform Services and their Abstraction

A defined **distinction between application and platform functionality** is needed during early platform evaluation. The application itself must be platform independent, merely capturing the requirements in terms of computation, communication, and memory accesses. At the same time, services and functions provided by target platforms may not be available at an early exploration stage, or they could be subject to the exploration itself, e.g., if they depend on the memory hierarchy or the chosen interprocessor communication scheme. The impact of such functions is exemplarily discussed in Section 6.6.3. In the context of performance simulation, such services may in addition be implemented differently on the simulation host for performance reasons or convenience. For example, native memory management, default packet libraries, or the SystemC scheduler could be used. Aspects of this have been discussed very recently in [85].

Therefore, we propose the following common functions that have the biggest impact on system performance to be provided as ***platform services*** during application modeling. This hides their implementation and allows to abstract their performance during exploration. A robust interface is given by code macros that hide these functions during profiling and back-annotate fixed and dynamic costs during simulation.

- *Memory Management* – the dynamic allocation of memory blocks through `MALLOC`, `AREA_MALLOC`, and `FREE`. Performance depends on the allocation algorithm, buffering strategy, memory hierarchy and mapping, and system state.

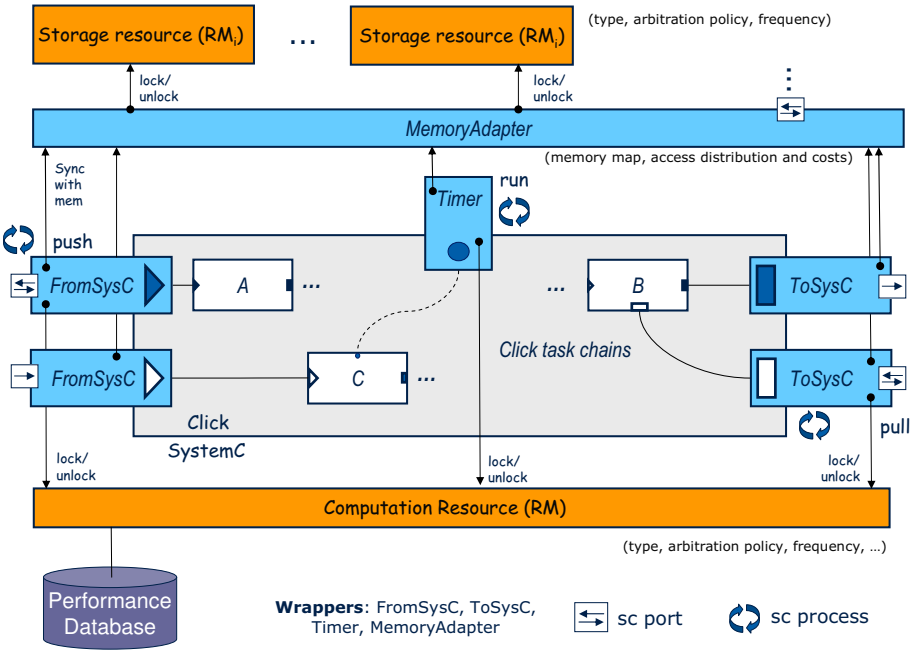



***Figure 5.6*** *– Performance model for tasks mapped to a processing element (represented by the computation RM), SystemC wrappers (in blue) including an adapter to associated storages.*

- *Scheduling and Packet IO* – activation of tasks through timers and IO. `SCHEDULE` and `UNSCHEDULE` abstract both fixed and dynamic parts that, e.g., take the current number of tasks into account. Packet descriptors are made available to the PE, and a `HEADER` function caches the packet header locally.
- *Token communication* – communication outside the packet flow. `Tokens` are available locally and in between PEs. They can be generated and copied at fixed costs.
- *Standard packet functions* – functions provided by Click, including modifiers such as `PUSH/PULL` and functions for packet creation/deletion. Expensive cases, e.g., requiring reallocation need to be handled explicitly by the designer.
- *Extended packet functions* – functions to join, split, modify, and efficiently create and kill packets, e.g., for (de-)aggregation. Actual costs highly depend on the chosen implementation and mapping. Optimizing the standard function, packet `CLONE`, e.g., for retransmission can be performed implicitly by IO interfaces.

### 5.4.3 Performance Simulation and Instrumentation

Performance simulation executes task chains explicitly within a concurrent SystemC environment that is provided by wrappers. Wrappers (cf. Fig. 5.6) handle a task chain's communication, interface it to the shared computation and storage resources, and synchronize its processing time with the simulation time. Using an XML-based performance database, each task is associated with costs for a particular resource, e.g., its processing time and accesses to memories (see also Sec. 5.5.2). At runtime, these costs are accumulated at *instrumentation*

*points* from *performance tags* stored in the database. Depending on the simulation mode, instrumentation points can seamlessly be used as additional *synchronization points* with the SystemC environment. This does not impact the accuracy of the software profile but merely the distribution of memory accesses and the resolution of the analysis. Thus – in the other extreme – synchronization may only take place at points of IO to reduce SystemC events for better simulation performance.

The same instrumentation points are used for both tagging and synchronization during simulation and for reporting instruction counts during profiling. They are added as macros to the element source code, as explained in Section 5.6. A typical Click element has three to five of such points. Leveraging Click, natural points for instrumentation can be found at element boundaries such as push/pull calls and function returns. Platform functions have their own instrumentation points. Additional *update points* can be added to increase accuracy, down to basic blocks. For convenience, points can also be tagged with static costs. A runtime parameter may be used to efficiently capture dynamic data and system dependencies, e.g., on packet length, in loops, or on the number of active clients.

### 5.4.4 Profiling Approaches

Two profiling approaches are presented, an extension to the conventional gnu profiler *gprof* and a new and automated approach that exploits SystemClick's instrumentation macros and that is able to capture memory accesses in addition to performance. Both approaches rely on the *MIPS-Sim Instruction Set Simulator (ISS)* [160], but are also compatible with other simulators. Click models are mapped to a single-processor target using SystemClick's code generation framework CRACC [210] and the MIPS-SDE tool chain [161]. The simulator is run for a specified time in instruction- or cycle-precise mode with caching disabled. Mapping onto a single core includes the use of a central scheduler – the lightweight NOVA scheduler – and may necessitate the inclusion of queue elements to decouple the model. However, the function of single elements is not influenced.

#### Conventional Profiling with gprof

This straight-forward approach for performance analysis at the granularity of elements extends on the tool ***gprof***. Gprof relies on built-in performance counters and coarsely instruments code to produce a function call graph, which necessitates two separate simulation runs to reduce profiling overhead. The application model is cut to subgraphs in order to expose relevant processing paths manually. Stimulation and measurements are aided by a profiling-only element *GMonControl* that allows dynamic activation of measurements during simulation. Still, the overall process is tedious. Statistics per function are processed by tailor-made scripts for per-element breakdowns and subtotals of stacked functions as in Figure 5.7. Cycles counts are averaged over all activations. Cycles of subfunctions that are shared between elements can only be averaged, as the call stack on MIPS architectures is not easily available. This leads to further inaccuracies.

```
SetCRC32 (1x)
-SetCRC32_simple_action                 |     3700/  100 =   37
  -Packet_length                        |     2400/  600 =   4.0  .  ( 100/ 600 x   4.0 =   400 )
  -Packet_data                          |     2600/ 3700 =   0.7  .  ( 100/3700 x   0.7 =    70 )
  -update_crc                           |   115102/  200 = 575.5  .  ( 100/ 200 x 575.5 = 57551 )
    -gen_crc_table                      |           ****
  -Packet_put                           |     2600/  100 =  26.0  .  ( 100/ 100 x  26.0 =  2600 )
    -Packet_tailroom                    |      400/  100 =   4.0  .  ( 100/ 100 x   4.0 =   400 )
    -Packet_uniqueify                   |     1000/  500 =   2.0  .  ( 100/ 500 x   2.0 =   200 )
  *** Packet_put = (32)
  -Packet_data                          |     2600/ 3700 =   0.7  .  ( 100/3700 x   0.7 =    70 )
                                    Sum |                  649.9  .  ( = 64991 sum total )
```

***Figure 5.7*** *– Output of the* gprof*-based profiling exposing cycle counts averaged over element execution and subfunction call graph.*

### Automated In-Place Profiling

The callgraph-based approach lacks automation, accuracy, and memory access profiling. Without changing or extending the simulator tool, three **other solutions** are available:

1. Use *instrumentation points* to report cycle counts measured during simulation. This is similar to gprof but allows for a finer granularity and automated back-annotation.

2. Analyze the *simulator trace* to extract and back-annotate memory accesses related to every instrumentation point and categorize them.

3. *Fine-granular instrumentation* of every instruction or every memory access. We discard this approach as discussed in Section 5.4.6.

**Our profiling method** [4] flexibly combines approaches 1 and 2, exploiting source code instrumentation as sketched in Figure 5.8. For approach 1, instrumentation points generate a trace of performance tags in a file during ISS, as shown on the left-hand side of the figure. On first invocation, every instrumentation macro dumps a unique id, and the profiling overhead to be subtracted from later measurements is calculated by measuring an empty block of code. During profiling, the macros read built-in performance counters, dump the result and their point id, and then reset the counter. Again, a run-time parameter, i.e., multiplier can be used to capture control and data dependencies more efficiently. For protection of platform functions, a `CONST_COST` macro is provided that excludes enclosed code from profiling or assigns a constant cost.

For memory access profiling (approach 2), the ISS-generated trace must be analyzed as shown on the right-hand side of the figure. Magic addresses are now used instead of dumping to mark instrumentation points. Post-processing scripts extract the addresses of read/write accesses and categorize them into predefined memory bins, stack, and heap regions (cf. Sec. 5.5.2). This speeds up subsequent performance simulations, since no address decoding is needed at runtime. Alternatively, traces of memory accesses can be stored for every instrumentation point and replayed during simulation.

Finally, cycle counts and memory accesses are accumulated to performance tags to be stored in the database. They can be calculated and analyzed, e.g., in the average or the worst case. A large mean deviation at a given point indicates that the current instrumentation does not properly capture the flow of control and should thus be refined.

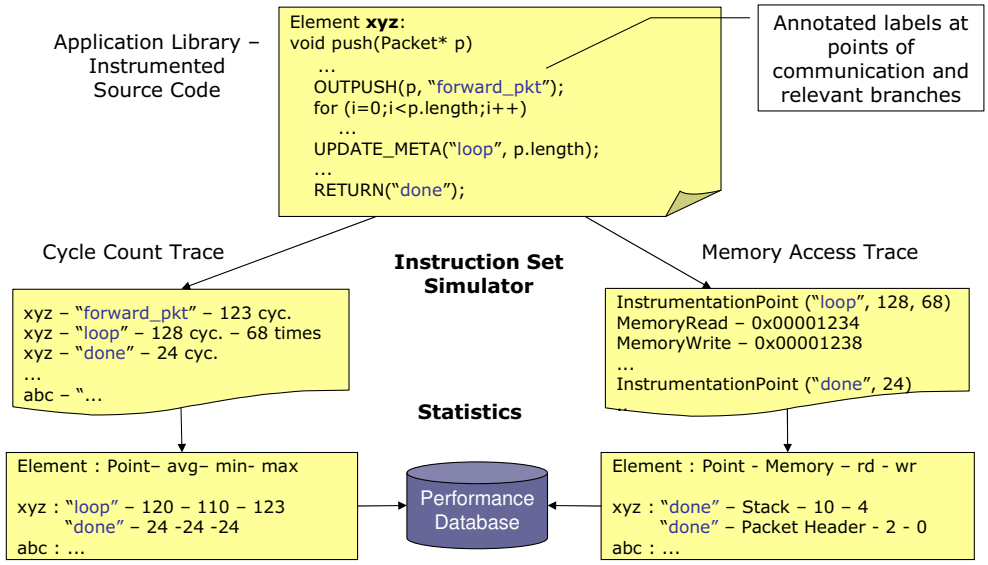



***Figure 5.8*** *– Automated profiling using in-place annotations and the simulation trace.*

### 5.4.5 Analysis

Performance analysis but also **visualization** of the application behavior is important for complex protocols such as WLAN. The introduction of instrumentation points at a user-defined granularity helps to understand the application better and faster. We leverage this abstraction by extending SystemC's tracing mechanisms to keep track of strings provided by ResourceManagers (RMs) that indicate the execution of tasks and instrumentation points. In addition, arbitrary variables can be traced, e.g., state in Click elements, filling level of queues, or memory utilizations. Since tracing only updates when synchronized with SystemC, additional synchronization points in the source code can be used for debugging (cf., Sec. 5.4.3). An example trace visualized in a trace viewer is shown in Figure 5.9.

In addition, a range of **statistics and extended functions** are provided by SystemClick to support the quantitative evaluation and verification of design points:

- *ResourceManagers* gather data on utilization in addition to tracing. They report the overall idle and service time distribution as well as per-task breakdowns of waiting and service time (min, ave, max). RMs could also be used for power analysis.

- *IO elements* gather statistics on packet flows broken down to packet size/type. They report packet numbers and (min, ave, max) processing time per length/type.

- *Packets* can store the sequence of passed elements over their lifetime. Time stamps for creation and deletion are annotated for analysis of delay and missed deadlines.

- Additional *Click elements* can be used to verify protocol conformance, measure throughput, and monitor, e.g., response timing.

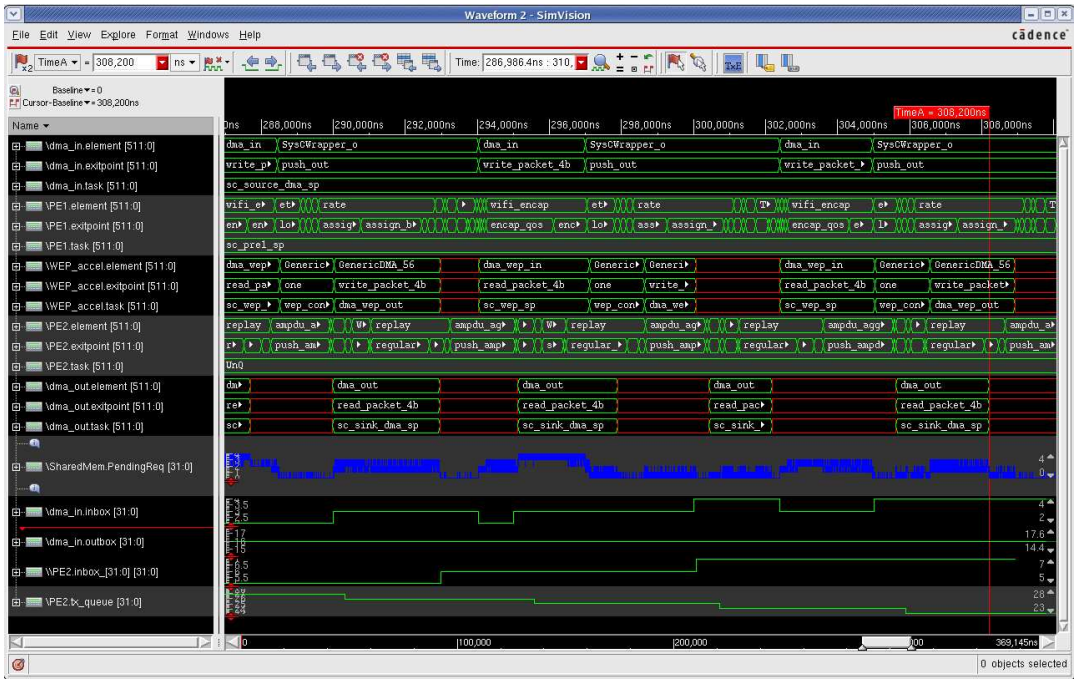



***Figure 5.9*** *– SystemClick trace file in* Cadence SimVision. *In addition to element execution traces (upper part), the number of pending request at the shared memory – indicating its utilization – and filling levels of queues are shown (bottom part).*

### 5.4.6 Discussion and Related Work

Our approach is both **flexible and fast** as needed for early exploration of the WLAN application on a programmable platform. Task chains remain explicit in SystemClick such that they can be flexibly assigned to architectural resources. Still, speedups of at least four orders of magnitude over RTL simulation are achieved while accuracy remains within 5 %, as is discussed in Section 5.6. At the same time, SystemClick allows verification of real-time processing paths and therefore of architectural feasibility, as the focus can be increased and worst-case performance can be considered if needed. The generated performance tags/points representing relatively large intervals together with pre-categorization of memory accesses are the basis for fast memory models as explained in Section 5.5. In addition, they provide an abstraction above the C language that eases the consideration of potential optimizations and allows productive analysis and debugging. A typical drawback of simulation-based profiling is that code related to exceptional behavior can be missed during initial measurements. However, exploration of packet processing applications typically focuses around steady-state behavior, and unknown performance tags are reported and can be given manually if necessary.

The profiling methods offer **trade-offs** targeting different points in the exploration and development cycle:

- *GProf-based* profiling allows an early understanding of costs and dependencies in the application. However, it is less suited for complex behavior and lacks automation and memory access profiling.

- *Inplace dump* is automated, fast, and captures computation cycles accurately at a user-defined granularity. It still lacks memory profiling, but can be used for fast update cycles during development.

- The exhaustive *trace-based approach* captures computation cycles and extracts and pre-categorizes memory accesses. It is slower, since the trace must be stored (see, e.g., [157] for a discussion) and analyzed.

A broad range of approaches to performance evaluation and profiling exists. As discussed in Section 5.2.3, we focus on **simulation-based methods that execute the system function natively** on the simulation host and back-annotate estimated or profiled performance. These can be distinguished by their annotation granularity, where our approach combines advantages of both abstract and accurate methods.

**Abstract / high-level performance annotation** frameworks target specifically early exploration and use very coarse-grained annotations. Simple approaches manually insert wait statements [28]. An important aspect is flexibility of the given annotations. We agree with Kogel [111], who demands that function and performance should be orthogonal at an early stage of development. Another aspect is high simulation performance as needed for RTOS-based SW development [250]. In this context, Schirner [212] discusses the importance of synchronization granularity and annotates ISS-measured performance at the function level. In the *SESAME* framework [186] a model calibration technique is proposed and simulation can be accelerated by abstracting, e.g., loops based on dynamic parameters, similar to our work. Only the *MESH* framework has been extended to model memory accesses efficiently [30]. However, all approaches are lacking performance feedback accurate enough for real-time applications.

**Fine-granular performance annotation** approaches aim at maximum accuracy, but vary in the granularity of the performance analysis – from binary level to single lines of source code – and performance annotation – from binary level to basic blocks. They replace slow, interpretation-based ISS and compile performance annotations statically into the host executable instead. Binary-level approaches estimate the costs of every instruction, as in [20]. An expensive memory model is needed as the target address space is mapped directly to the host. Kempf [106] relies on a partly-optimized three-address intermediate representation (IR) generated by the *Lance* compiler [132] for fine-granular annotation of static costs and explicit instrumentation of memory accesses. This approach can be extended with ISS for increased accuracy in critical parts [64]. Annotation on a line-to-line basis in the source code as proposed in [157, 214] complicates consideration of optimized code. Overcoming some of the problems of above approaches, Wang [240] uses both static and dynamic methods at various abstractions for analysis and annotates performance up to the level of basic blocks. Like Wang, Cheung [42, 43] considers aggregation of atomic wait statements for improved simulation performance.

Fine-granular processor models are often used in virtual prototypes, as they allow full and precise software coverage. This may be necessary, e.g., for verification of large-scale systems. In our application, few control-dominated elements such as EDCA would benefit from a complete coverage of basic blocks. However, their performance impact is limited. In addition, during development and exploration, preciseness is often not needed, or even misleading, as many parts of the system are still work in progress. Even worse, profiling of larger chunks of code as in our approach proved better suited for reflecting dynamic processor effects and compiler optimizations. SystemClick avoids the following **drawbacks**

of most fine-granular approaches that slow down simulations and hinder use for early exploration and fast development:

- *Lack of abstraction* – No abstraction above the (C) language level is provided, e.g., for analysis and performance re-evaluation. No abstraction of data dependencies and grouping or categorization of memory accesses is available to speed up simulation. This is further discussed in Section 5.5.5.
- *Lack of flexibility* – The application timing cannot be changed dynamically for exploration. Platform functions cannot be considered separately. Tasks are regarded as SW and cannot be partitioned or assigned to different resources for exploration.
- *Tool infrastructure* – Target-specific compilers or simulators need to be modified, which may be tedious or impossible for commercial tools.

## 5.5 Memory Exploration in SystemClick

The memory hierarchy of an application-specific embedded system significantly impacts its resulting performance and cost [178]. In fact, the choice of the memory hierarchy can have a bigger impact than the choice of the processor core [165]. Thus, its key properties and the interplay with the application must be explored early in the design process. Click's model of computation makes the application's function independent of memory interference, since all interactions are explicit. However, especially shared memories introduce access and thus performance penalties that are difficult to estimate due to the effects of latency, arbitration, and multiple clock synchronization.

SystemClick effectively decouples simulation execution from performance estimation, which makes explicit modeling of the memory hierarchy necessary. Memory models are thus needed that reflect access penalties as precisely as possible. These models must be fast since vast numbers of design points must be evaluated due to multiple design axes and large parameter ranges. We have extended SystemClick for the exploration of memory hierarchies [5]. The modified framework targets the embedded memory design space (Sec. 5.5.1), leverages SystemClick concepts (Sec. 5.5.2), and incorporates memory models at different abstraction levels (Sec. 5.5.3).

### 5.5.1 The Embedded Memory Design Space

We are interested in the exploration of memory hierarchies for programmable SoCs as they are used for protocol processing in devices such as home gateways and network processors. Such **deeply embedded systems** exhibit specific characteristics to fulfill their performance and real-time requirements:

- *Processing elements (PEs)* deploy small RISC-like processor cores. The cores have simple pipelines and do not support, e.g., out-of-order processing. Their operating frequency varies with the application. The local subsystems comprise instruction and data memories of variable size running at core speed. Caches are avoided due to their unpredictable timing behavior. Local memory content is loaded explicitly.

- The *number of PEs* ranges from one to the 10s. Each PE may have an individual configuration of type and frequency, local storage, and communication and memory ports. Inter-PE communication (e.g., message passing) may take place directly or via shared memory.
- *Shared memories* are primarily SRAM and are kept on-chip for cost and performance reasons. They may be organized in multiple banks and are accessed via multiplexed bus/crossbar structures. There operating frequency is different from the PE frequencies. Thus, accesses require synchronization in addition to arbitration.
- *Other clients* such as on-chip CPUs or DMA engines may access selected shared memories as well, effectively increasing the contention and limiting the bandwidth.

Shared memories are typically accessed through a shared bus or a cross-bar. We distinguish the local (e.g., PE) clock and the clock of the access system. In both cases, the overall **timing behavior** of memory accesses (see also Sec. 7.3.3) is **influenced** by:

- *Synchronization* – The local clock needs to be synchronized with the memory subsystem, for example, through an asynchronous FIFO. This typically has a static part (FIFO delay) and a dynamic part (clock synchronization of the faster clock). For read requests, the effect also occurs for responses.
- *Arbitration* – The identification of the next client to be granted access to a memory typically takes one clock cycle (of the access system) and is based, e.g., on FCFS, Round-Robbin, or priority schemes.
- *Memory Latency* – For read requests, the round trip delay of the memory becomes important. If the memory system is not fully pipelined, it can be blocked by previous requests. If split transactions are not supported, the communication system is essentially blocked during a read request.

### 5.5.2 Memories in SystemClick

**Representation and mapping of data and storage.** Data objects in the functional model are described using regular C/C++ constructs. Memory areas for common heap, stack, and compile-time data sections are created by default. The user may add extra heaps or compile-time sections. Distinct data objects such as lookup tables can, e.g., be represented and mapped with place-holder Click elements (cf. Sec. 5.3.1). SystemClick provides a special *AREA_MALLOC* instruction to allocate and tag objects with a memory ID, enabling automated access profiling (cf. Sec. 5.4.4). Depending on the desired mapping and analysis resolution, a memory ID may identify individual objects or groups of objects to be allocated into the same memory area. In the mapping phase, each memory ID is associated with a storage resource of the platform. Using a *memory map*, every computational resource may have its own mapping and access parameters.

**Performance simulation.** Tasks are executed in SystemC by wrappers (cf. Sec. 5.4.3). This necessitates explicit consideration of memory access penalties. The performance database holds tags with the number of accesses to different – pre-categorized – memory areas that are accumulated at simulation runtime at instrumentation points. Such tags can be derived, e.g., by automated profiling (cf. Sec. 5.4.4). Depending on the simulation mode, the granularity of synchronization points and thus the length of the interval to be synchronized can be chosen seamlessly from instrumentation points. Accumulated memory accesses within such an

interval may be handled individually (single access mode) or as a group (grouped access mode) by a *memory adapter*. The adapter handles accesses per storage resource to derive memory contention and access penalties. Alternatively, precise memory access traces can be replayed during simulation. Due to pre-categorization and direct indexing, no memory addresses need to be evaluated at run-time.

**Memory hierarchy analysis.** ResourceManagers trace idle/service utilization per task. Building on these statistics and trace capabilities, specific performance data are gathered on 1) the accesses and stalls per processing resource and memory area, and 2) on the back log and waiting times at arbiters. Combined with other performance results such as throughput, response delay, dynamic memory usage, and precise execution traces (cf. Sec. 5.4.5), these data are the input for analyses as shown in Chapter 6.

### 5.5.3 Memory Access Models

In SystemClick, the accumulated computation time $\mu_{\text{comp}}$ and pre-categorized memory accesses at a synchronization point are known prior to the actual synchronization with simulation time. After associating memory area accesses to storage $mem_i$ using a memory map associated with every computational resource, the tuple

$$\mu = (\mu_{\text{comp}}, \{\mu_{\text{mem,i}}\}), \qquad \mu_{\text{mem,i}} = (\text{acc}_{\text{i,read}}, \text{acc}_{\text{i,write}})$$

is presented to the memory adapter for synchronization.

This allows different approaches to reflect the latency of shared memory accesses: Accesses can be issued one-by-one to a representation of the architecture. Or, all accesses for the current interval can be grouped in order to predict additional contention penalties, e.g., on a statistical basis. Finally, access latencies to selected storages can already be considered statically in the computation time $\mu_{\text{comp}}$ to speed up simulations.

#### Single-Access Mode

In single access mode, the **load/store requests** for all memories $acc_{total}$ are ordered by the memory adapter (MA), e.g., using a Deficit-Round-Robin schedule to guarantee an even distribution. Every single access is issued either to an abstract resource manager $RM_i$ representing storage $mem_i$ or passed to the SystemC environment, for example, to a TLM model (cf. Fig. 5.6). The arbitration schemes at the $RM_i$ can be customized like all ResourceManagers in SystemClick. Schemes include e.g., Round-Robbin or FCFS.

Each access is divided into **five steps** by the MemoryAdapter, where steps 1 to 3 can be replaced by a single call to a SystemC port:

1. Call the blocking *lock()* function of $RM_i$
2. *wait()* for the arbitration period of $RM_i$
3. Call *unlock()* function of $RM_i$
4. Account for extra delay associated with $mem_i$
5. Account for a fraction of the total computation time

Arbitration period and extra delay can be resource- and load/store-specific and may consist of local PE and memory cycles. See Figure 5.10 for an exemplary read request.

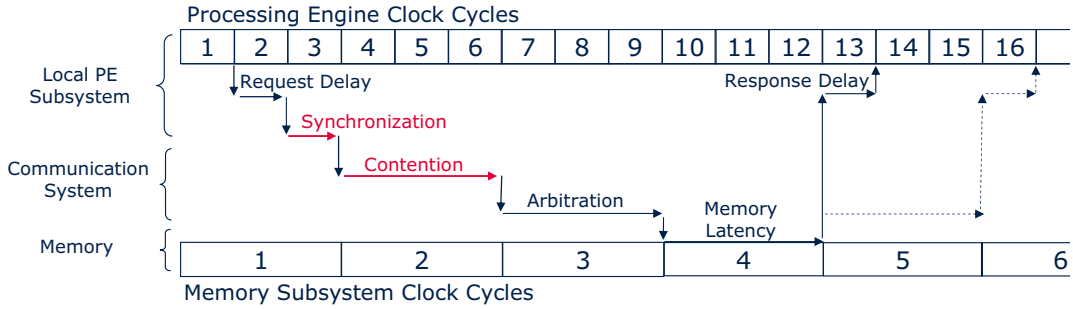



***Figure 5.10*** *– Read request to a shared memory. Dynamic effects caused by synchronization and contention need to be considered in addition to static latencies and delays.*

We implement a number of random, fixed, and trace-based **distributions** in the MA to determine the proportion of $\mu_{\text{comp}}$ in between two subsequent accesses. For instance, a common approach in literature is to assume a negative exponential distribution, where we choose the parameter $\lambda = \mu_{\text{comp}}/acc_{total}$. In order to guarantee the total bound given by $\mu_{\text{comp}}$, the rounding of cycles is balanced according to an updated $\lambda$, and accesses are squeezed if the end of the interval is approaching. As another example, access traces derived from automated profiling can be used for a most accurate distribution.

For cycle-accurate consideration of synchronization effects between differing local and memory clocks, it is necessary that the $RM_i$ representing storages have a notion of time. For this, the $RM_i$ defer the arbitration of the next client to the SystemC *update()* phase at the end of their current clock cycle. This synchronization of *clocked RMs* is demand driven and only necessary when asynchronous requests are present.

### Grouped-Access Mode

In this mode, **all accesses** to given storages $mem_i$ during an interval $I = \mu_{\text{comp}}$ are **grouped together** and evaluated by newly introduced statistical $RM_i$'s. Latencies for memory accesses are abstracted in the computation time $\mu_{\text{comp}}$ as accumulated by performance tags. The effect of clock synchronization between PEs and memories is considered statically, e.g., in the average or worst case. However, delay caused by memory contention is reflected in one (as in [30]) or more penalty intervals following the initial interval $I$ (see Fig. 5.11). With prerequisites as above, we execute four phases at a synchronization point:

1. Set client request rate $\rho_j = \mu_{\text{mem,i}}/I$ and interval duration I at all $RM_i$
2. *wait()* the interval time $I$
3. Enter penalty phase: Repeatedly retrieve and *wait()* penalties $\Pi_i$ until remaining penalty is below threshold
4. Clear client request rate at $RM_i$ if no penalty left

Statistical $RM_i$ keep track of their **utilization** $\rho$, the set of active clients $C_{act}$, and requested utilizations $\rho_j$ of every active client $c_j \in C_{act}$. Only when request rates are set, cleared, or penalties retrieved, the state is updated by calling an `updatePenalties()` function that computes penalties for the passed interval $I$ a posteriori, e.g., using a statistical contention function. Of course, this assumes memory accesses and requests served to be equally distributed in any given subinterval. Setting the contention function to zero, only memory latencies are simulated, yielding lower bound execution times.

During the **penalty phase**, clients iteratively consume, i.e., `delete()` and `wait()`, their current penalty $\Pi_j$ without clearing their requested utilization: This allows to consider contention for other clients caused by accesses deferred into the penalty phase. Without the iteration in the penalty phase, the approach of Bobrek [30] is covered, which ignores contention during penalty intervals.

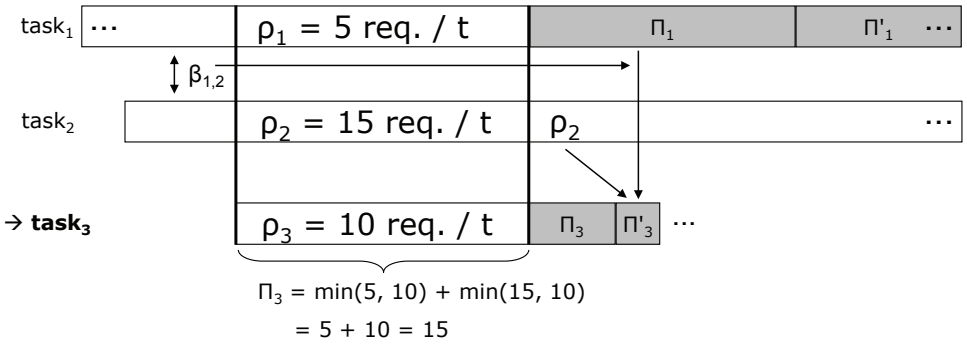



***Figure 5.11** – Example of estimating contention during an interval I in grouped mode. Penalty phases $\Pi_j$ account for additional latency due to delayed accesses.*

### Iterated Contention Algorithm

In order to derive strict upper bounds for contention, iteration is needed as additional contention might occur during the penalty phase. In this section we provide a memory model suited for grouped access mode, which is able to estimate the total contention that can possibly occur, i.e., in the worst case. The scheduler is arbitrary but assumed to be fair, i.e., every client $c_j$ is served at least $1/n_{act,J}$ of its request utilization $\rho_j$ during each interval $J$, where $n_{act,J} = |C_{act,J}|$ denotes the number of active clients during that interval. The case of prioritized schedulers is well understood and out of the focus of this thesis.

In a **first approach (S)**, we exploit scheduler fairness to derive the number of totally outstanding requests $\alpha_j$ of client $c_j$ in the worst case: Initially, $\alpha_j$ is $\rho_j \times I$, where $I$ is the interval (length) if no contention occured. For any subsequent or sub interval $J$, $\alpha_j$ can be reduced by at least $\rho_j \times J \times (1/n_{act,J})$. Now, the penalty $\Pi_j$ equals the number of outstanding requests $\alpha_j$ and has to be recomputed for every subsequent interval. The approach, however, leads to overly pessimistic bounds and performs badly due to many iterations of update intervals.

For an **improved approach (P)**, we first observe that the worst-case penalty incurred by any client $c_j \in C_{act,J}$ in any interval $J$ is given by

$$\Pi_j = \Big( \sum_{c_k \in C_{act,J}, c_k \neq c_j} \min(\rho_j, \rho_k) \Big) \times J \,. \tag{5.1}$$

```
for all active clients c_j ∈ C_act
 update outstanding requests α_j              /* as in (S) */
 if (no accesses left)
  set Π_j to 0 and continue

 /* Compute penalty π_j,k between c_j and c_k */
 for all other clients c_k
  set π_j,k to min(ρ_j, ρ_k) × J               /* Eq. (5.1) */
  if (c_k is already in penalty phase)
   bound π_j,k by a priori penalty β_j,k of c_k
   subtract π_j,k from β_j,k
  add π_j,k to Π_j

 bound Π_j by outstanding requests α_j        /* as in (S) */
```

***Figure 5.12*** *– Function to update penalties $\Pi_j$ of all active clients $c_j$ for interval $J$ in the combined iterated worst-case contention algorithm (P).*

Any request made by $c_j$ in $J$ can only collide at most once with every other client, again, using scheduler fairness (see also Fig. 5.11).

It is important to note that all possible contention between two clients $c_j$ and $c_k$ is already considered prior to their penalty phases. Thus, the penalty at $c_j$ incurred during $c_k$'s penalty phase cannot be larger than the penalty $c_k$ already had when $c_j$ became active. In this case, we can bound the partial penalty $\pi_{j,k}$ by the *a priori* penalty $\beta_{j,k}$, which is stored for all other clients $c_k$ whenever a client $c_j$ becomes active.

Both approaches are independent and can be **combined**, as sketched in Figure 5.12: The penalty is computed using Equation 5.1 and bound by both a priori penalty and remaining accesses. The worst-case penalty derived by the presented algorithm is a good indication of the actual amount of contention that may occur. For an estimate of the average case, this penalty can be scaled in each interval $J$, e.g., based on $|C_{act,J}|$, total utilization $\rho$, $\rho_j$, or the burstiness of the client. The *grouped average (grouped avg.)* approach used for the evaluation in the next section divides the penalty by the number of active clients $|C_{act,J}|$ in any given interval. This effectively distributes contention evenly amongst the clients.

### 5.5.4 Evaluation of Memory Models

We use the transmission path of the IEEE 802.11n application (cf. Chap. 3) to assess and compare the models. The system function is mapped to two PEs (cf. Sec. 5.5.1) running at 200 MHz. Including three DMAs for packet IO and an encryption accelerator, five clients access a single shared memory through a bus. The performance bottleneck is at the first PE, for which the packet throughput in Mbit/s is stated in subsequent figures. A detailed analysis of accuracy and simulation performance follows in Section 5.6.

#### Impact of Access Distributions

In literature, memory accesses issued by processing cores are commonly assumed to be exponentially distributed. We confirm that this assumption is also valid for our system, assessing

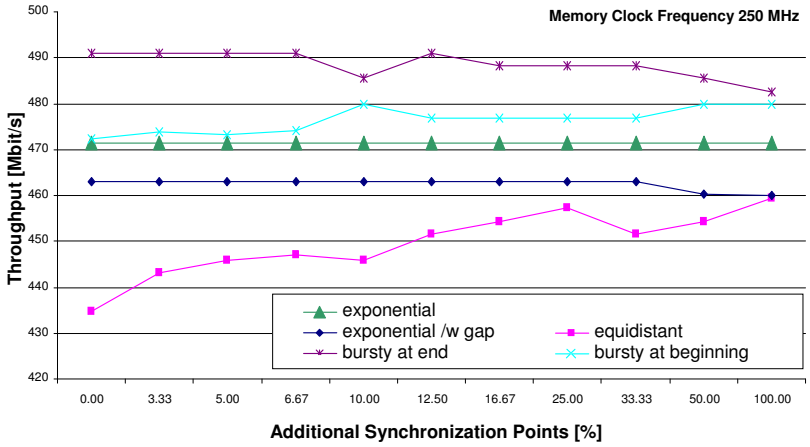



***Figure 5.13*** *– Impact of access distribution and synchronization granularity.*

the sensitivity of system performance to access distributions and simulation resolution in single access mode.

Memory arbitration is based on *Round-Robin* for this study. Because back-to-back accesses will always incur a maximal waiting time, a **distribution's burstiness** influences the amount of contention. We consider two instances of two corner cases, respectively:

- *All bursty access* – Two patterns where all accesses are performed as a block, either at the beginning or at the end of the interval.
- *Access with inbetween gaps* – If possible, accesses are never performed back-to-back. For one pattern, accesses are distributed equidistantly, for another pattern, accesses are randomly distributed but a gap is enforced in between accesses.

Now, these corner cases are **compared** with the truly random exponential distribution as simulation resolution is increased. This increase is achieved by using additional synchronization points (cf. Sec. 5.4.3): At 0 %, accesses are averaged over complete tasks. At 100 %, up to 60 synchronization points per task account for around 100 cycles and 25 accesses each. As shown in Figure 5.13, the considered variants of the two corner cases converge with increasing resolution, respectively. Over 12 considered memory frequencies, the average spread from their mean decreases from 2.4 % to 0.4 % for bursty and from 1.6 % to 0.8 % for gap patterns. This indicates that the structure within the variants becomes almost irrelevant in the highest simulation resolution. The exponential pattern remains nearly constant for all synchronization/simulation resolutions.

Over all distributions, the exponential pattern deviates only 0.5 % from the mean of the two corner cases. Compared to simulation based on a cycle-accurate access trace, it deviates only by 1.5 %. Thus, the exponential distribution is a good approximation for our system, even at a low simulation resolution, and will serve as our reference.

### Comparison of Access Models

A comparison of single and grouped access models in terms of system throughput over memory clock frequency is shown in Figure 5.14. We consider a set of 12 frequencies that

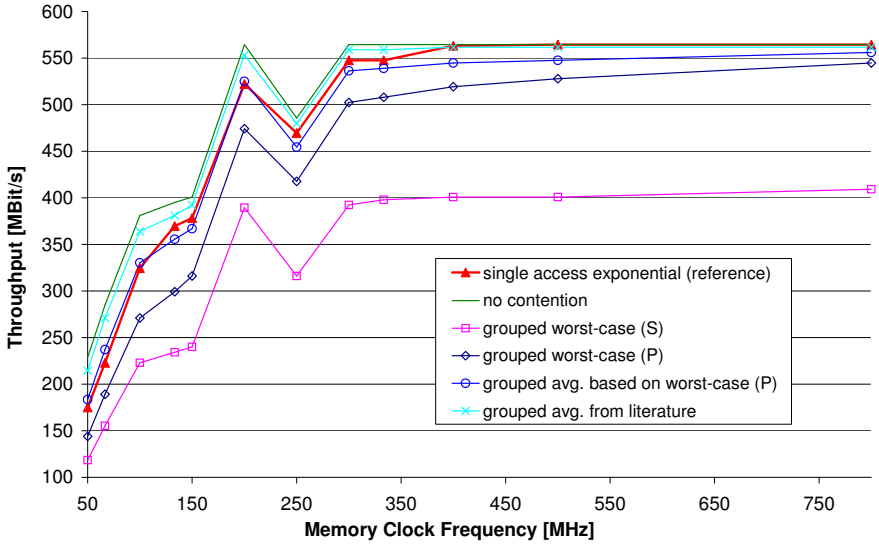



***Figure 5.14*** *– Comparison of single and grouped access models.*

can be derived by a natural fraction of the PEs' clocks. Still, clock synchronization effects limit the throughput at 133.3, 150, and 250 MHz. This is also well reflected statically in the grouped models. For the single access exponential model, which is used as a reference, the penalty caused by memory contention is up to 23 % at 50 MHz, drops to 7.5 % at 200 MHz, and vanishes with very fast memories. The iterated worst-case algorithm (P) outperforms the simple approach (S) and predicts a throughput 13 % below the reference on average. Based on this worst-case contention, the *grouped avg.* approach estimates the actual contention by dividing the penalty by the number of active clients in every interval. It stays within 2.6 % of the reference throughput in the range up to 333 MHz, up to four times better than the approach [30] based on [40], which underestimates contention greatly. See Section 5.6.2 for a comparison with cycle-accurate simulation.

### 5.5.5 Discussion and Related Work

The proposed models in combination with performance feedback (cf. Sec. 5.4) allow both efficient evaluation of penalties for single design points as well as fast exploration of memory mappings and architecture characteristics. It is well suited for the domain of deeply embedded systems and offers trade-offs that target especially early exploration but also allow verification of real-time constrains. In **summary**:

- *Trade-offs* – The synchronization granularity impacting the distribution of memory accesses can be varied. In the extreme, single-access mode or access traces can be used for verification purposes.
- *Fast Simulation* – The chosen abstraction and the statistical models allow consideration of accesses over large intervals, reducing simulation overhead. Pre-categorization into memory areas saves expensive run-time parsing of addresses.
- *Fast exploration* – Mappings of data structures can be changed efficiently, and the architecture model is sufficiently abstract to be changed quickly.

Some issues should be addressed in future work. If clients access multiple shared resources, accesses could block each other, which is currently not fully reflected in the grouped-access model. For single-access simulation, more complex architectures can be constructed from standard SystemC components and SystemClick's ResourceManagers. In addition, the iterated worst-case algorithm (P) is not optimal in all cases, being too pessimistic, e.g., if multiple clients request 100 % utilization.

Our exploration approach can be **classified as hybrid**, as it combines simulation with analytical methods and combinatorial arguments to estimate contention in given intervals. We thus discuss analytical, simulation-based, and hybrid methods, and memory exploration in general. But none of the approaches is suited for our design space, reflects complex application behavior, and allows fast early exploration at the same time:

**Analytical models** for the *memory* or *machine interference* problem based on queuing or Petri networks have been used to reflect the impact of memory arbitration. They barely reflect the control flow and are tailored large-scale systems. Furthermore, they impose restrictions on distributions and rates of request. Early work targeting mainframe computers can be found, e.g., in [22]. The techniques in [25] include Markov chain models and analytical methods. Hoogendoorn [83] was one of the first to rely on a combinatorial approach. Newer bus-based and cached multiprocessor architectures were considered, e.g., in [164]. Mean value analysis is a popular method for solving queuing networks [197]. A recent bibliography on machine interference is found in [76, 228]. Memory is partitioned for generic applications based on analytical models in [94].

Most models are too complex for our purpose. A practical approach is found in [40], which we used as a reference in Section 5.5.4. There, a formula for the average response time $T_k$ of a memory request is derived, where $k$ is the number of concurrent clients:

$$t_k = \delta_S(1 + const(k)U_R)$$

This scales the access latency $\delta_S$ by the request utilization $U_R$ of the shared resource and by a constant depending on $k$. However, the model is inappropriate for small numbers of clients, as $const(k)$ becomes 0 for $k = 1, 2$ and does not reflect asymmetric load requests. When applied to our problem, contention is thus underestimated greatly.

**Simulation-based approaches** in general were discussed in Section 5.2.3. Even for architectural exploration there is an industry trend toward very accurate co-simulation (e.g., by *ARM, VAST*) that synchronizes every memory access with precise models, e.g., in TLM abstractions [211, 11]. Such models can vary the level of abstraction but are still detailed and tedious to change. Models are often biased and focus around bus systems such as *AMBA* in [180, 143]. Recently, the function has been abstracted completely to generate artificial memory traffic for complex bus models by *CoWare* [44].

Revisiting performance annotation frameworks (cf. Sec. 5.4.6), efficient memory abstractions are rarely found. Most fine-granular approaches [106, 42] and even abstract ones [111, 28] synchronize single transactions or ignore the problem [157]. Some consider the use of cache simulators to speed up simulations, e.g., [214, 240]. Problems include proper address generation [43] and fast modeling of cache strategies [66]. Wang [240] leverages the similarity to host access traces and accumulates accesses for basic blocks, which is more efficient. Unlike our pre-categorization approach, such cache models must, e.g., rely on binary search to determine cache contents. In addition, they focus around modeling cache strategies rather than the abstraction of arbitration effects.

**Hybrid simulation-analytical approaches** have been proposed by Bobrek [30]. Discrete event simulation is combined with statistical models to compute penalties for shared resource accesses, similar to grouped accesses in SystemClick. However, a proprietary scheduler is used instead of SystemC. Contention during penalty phases is ignored, and they use the imprecise model from [40] (cf. Sec. 5.5.3). In [29], this model is extended to statistical regression learning based on *access attributes* to predict contention. Average requested utilization $\rho$, access balance $B$, and number of active threads $T$ are identified as the most significant parameters. Our approach considers the same parameters, but details accesses for every client instead of generalizing the access balance. By learning functions $f_1, f_2, \beta$ from co-simulation, the delay per unit time (DPT) can be estimated as

$$\mathrm{DPT} = f_1(\rho) + f_2(B) + \beta T \, .$$

While results seem promising, a number of drawbacks exists: First, precise co-simulation of the complete system must be available. This is not the case early in the design process, and difficult if the memory subsystem itself is subject to exploration. Furthermore, a strong dependency on interaction patterns during the initial training period is reported. This is less severe for the considered homogeneous, loop-centric algorithms. By generalizing the access balance during initial training, however, application knowledge about heterogeneous task activation is lost, which is essential for WLAN. Finally, the authors find their approach less suitable for small numbers of clients.

Other hybrids include [213], which focuses on real-time behavior and derives worst-case timings from abstract stimuli. Shared resource accesses are grouped locally and then integrated into a formal system-level performance simulation. Similarly, the authors of [118] try to combine formal network calculus with simulation. Both approaches, however, focus on priority schedules and do not use an accurate description of the system's function.

**Exploration and optimization approaches** are often based on static methods that rely on an initial simulation and target loop-centric benchmarks that are easier to abstract. Dynamic and complex behavior as in WLAN thus cannot be reflected. For example, an abstract communication analysis graph (CAG) is extracted from co-simulation with Polis/Ptolemy in [119]. The CAG captures computation, memory accesses, and inter-task synchronization. After mapping, every transaction is evaluated in terms of access delay and contention. Statistical application graphs are used, e.g., in [65]. Approaches for scratch-pad and cache partitioning based on static application graphs include [176]. Static ILP formulations are used in [227] to derive scratch-pad allocation and static task schedules. Dynamic, compiler-directed allocation for heap memory to scratch-pads is shown to be more efficient than caches in [53]. In [177], generic memory optimization techniques are presented. There is a large body of work on loop optimization and memory design limited to algorithmic kernels, see, e.g., [236], and on cached systems [66].

## 5.6 Quality of Results

Simulation performance is essential, since the complexity WLAN and the large design space require a large number of long simulation runs. In addition, the results must be accurate enough for indicative evaluation of design points and verification of real-time constraints. The performance impact of architectural changes must be reflected properly. We present

optimizations to SystemClick, compare performance and accuracy to native X86 execution, ISS, and RTL simulation, and relate to other approaches.

### 5.6.1 Simulation Performance

First, we focus on the performance of SystemClick in standalone configuration, i.e., serving as a single-PE model comparable to an Instruction Set Simulator (ISS) embedded in SystemC. Previously [207], SystemClick's accuracy has been shown to remain within few percent in this case. However, the performance of the as-is implementation is insufficient for our purpose. Thus, we discuss optimizations and broaden the evaluation by assessing performance-influencing factors for different benchmarks.

**Framework Optimizations.** SystemClick's speedup over MIPS ISS is shown in Table 5.1 for the baseline implementation and a *WLAN-TX* benchmark as published in [9]. As absolute bounds, the benchmark is targeted to MIPS and X86 using CRACC (cf. Sec. 5.4.4). Native X86 simulation (A) shows a speedup of 4400X over ISS (D), which is in the range found in the MIPS documentation [161]. The upper limit for SystemClick's performance simulation is given by behavior-only simulation in SystemC (B), where the lightweight NOVA scheduler is replaced by the SystemC scheduler and SystemClick's wrappers encapsulate Click task chains. Running the benchmark on a processor model comparable to ISS necessitates resource arbitration, i.e., associating with a ResourceManager (C', untimed) and additional performance annotations (C, perf.). For the baseline implementation, we observe that behavior-only simulation reaches less than 50 % efficiency, affecting all other configurations accordingly. Furthermore, adding performance annotations limits the speedup significantly (C vs. C'). Consequently, we optimized:

- *SystemC infrastructure* – including the *lock()* procedure and event handling at resources and wrappers, and handling of empty resources. This increases the upper bound (B) by 1/3 to more than 60 % efficiency rel. to the native execution (A).
- *Performance annotations* – including caching of tags, optimized accumulation after initial execution, adjustable synchronization granularity, and better integration of tracing (as sketched in Fig. 5.15). This improves timed simulation (C) by 7X, approaching the limit given by (C') and only 58 % slower than native execution.

***Table 5.1*** *– Simulation speedup of SystemClick for the* WLAN-TX *preprocessing benchmark (baseline [9] and optimized implementation [4]).*

| | *Configuration* | *Baseline Impl. [9]* *Speedup[X]** | *Efficiency*** | *Optimized Impl. [4]* *Speedup[X]** | *Efficiency*** |
|---|---|---|---|---|---|
| A | Native X86, behavior only | 4400 | 100.0 % | 4400 | 100.0 % |
| B | SystemC, behavior only | 2070 | 47.0 % | 2750 | 62.5 % |
| C' | SClick /w untimed resource | 1500 | 33.9 % | 1870 | 42.5 % |
| **C** | **SClick /w perf. annotation** | 260 | 5.9 % | **1840** | **41.8 %** |
| E | SClick /w add. 2 PE arch. | 190 | 4.3 % | 511 | 11.6 % |
| E' | SClick /w add. shared com. | 180 | 4.1 % | 450 | 10.2 % |
| D | MIPSsim ISS | 1 | 0.02 % | 1 | 0.02 % |

*) Open-source SystemC kernel and commercial *MIPSsim* ISS **) simulated speedup / native speedup

***Table 5.2** – Simulation speedup over ISS for different benchmarks.*

| *Benchmark* | | WLAN-TX (Tab. 5.1) | WLAN-TX no CRC | WLAN-TX 2x CRC | PCI-X from [207] | headers (Sec. 5.6.2) |
|---|---|---|---|---|---|---|
| *Cycles / timing event* | | 5024 | 1050 | 9100 | 1200 | 583 |
| A | Native X86 | 4400 | 3650 | 4450 | 3970 | 3070 |
| B | SystemC behavior | 2750 (63%) | 1160 (32%) | 3270 (74%) | 1400 (35%) | 400 (13%) |
| **C** | **SClick perf. anno.** | **1840 (42%)** | **580 (16%)** | **2450 (55%)** | **680 (17%)** | **120 (4%)** |
| D | MIPSsim | 1 | 1 | 1 | 1 | 1 |

Open-source SystemC kernel and commercial MIPSsim ISS. Efficiency relative to native X86 execution given in brackets.

**Simulation Benchmarks.** The simulation performance is influenced by the characteristics of the benchmark. Especially event handling shows a large overhead in SystemC, whereas the lightweight NOVA scheduler can handle subsequent events much faster in ISS simulation. A main factor thus are the simulated *Cycles Per Timing Event (CPTE)*, which is related to simulation performance for a number of benchmarks in the following.

We analyze three benchmarks. First, the *WLAN-TX* benchmark is re-considered, which features protocol processing, timers, and payload processing such as CRC for a sequence of packets and proved representative for our domain. By doubling and removing the expensive CRC calculation, the CPTE is varied as shown in Table 5.2. The effect on performance simulation (C) is a drop by 68 % if the CPTE are reduced by 79 % or an increase by 33 % if the CPTE are almost doubled. Another benchmark from the packet processing domain, *PCI-X* [207], proves to be comparable to *no CRC*. Finally, the *headers* benchmark that will be used in Section 5.6.2 differs from the others, since it comprises little computation and many timers. It was designed specifically to evaluate extreme effects and accuracy of the memory models. In summary, we conjecture the **following influences** assuming that ISS (D) performs on a constant level.

- *X86 (A)* – Performs better for homogeneous processing (e.g., CRC) and worse for scheduling and control-dominated code, probably due to the superscalar and optimized host processor. In addition, caches may have an effect, e.g., on the *headers* benchmark, where a large number of packets is produced on store beforehand.

- *SystemC (B)* – In addition to event handling, performance can be impaired as native (libc) memory management is used rather than the NOVA implementation, which is very efficient in the absence of reallocations (e.g., for *headers*). Also, platform and packet functions are implemented differently.

- *SystemClick (C)* – Annotated simulation is influenced by all of the above factors. In addition, the granularity of the instrumentation (e.g., for very small elements as in *headers*) and the size of the element database can further impact performance.

For **typical packet processing applications**, a performance simulation speedup of almost three orders of magnitude over ISS and 20 to 50 % efficiency relative to native X86 thus are realistic. In fact, the sparse nature of the full WLAN application leads to even higher speedups in SystemC, since the discrete event scheduler skips over idle periods.

```
#define UPDATE_META_DP(name, label, multiplier)
  static void* caching_hook = 0;
  if (m->resource)
    update_meta_cached(name, m, label, multiplier, &caching_hook);
    if (SYNC_ELEMENT)
      if (COUNT_PP) m->pp_count++;
      rm_trace_annotation_point_d(name, label, m);
      wrapper_sync_simulation(m);
```

***Figure 5.15*** *– Optimized instrumentation macro for performance simulation with cached tags. Synchronization can be disabled or based on counters. The caching hook refers to a* MetaUpdate *structure that initializes the tag on first execution accessing the database and resource properties. Subsequent tag accumulation only requires direct indexing.*

### 5.6.2 Architectural Simulation

For full platform evaluation, the single-processor performance simulation models must be multiply instantiated and coupled with an architecture model. Thus, we extend the assessment to SystemClick's abstract architecture representation in the following. Table 5.1 showed the speedup for the WLAN-TX benchmark being mapped to a two processor system connected with FIFOs (E) or shared I/O boxes (E'). As communication channels and more ResourceManagers for shared resources and additional PEs are involved, the simulation performance is reduced by 72 and 76 %. Still, SystemClick runs approx. 500X faster than single-PE ISS simulation and at about 10 % efficiency compared to native execution while evaluating the full application on a concurrent architecture in SystemC.

In the following, we broaden this evaluation by reflecting a heterogeneous memory architecture in addition, using memory models as described in Section 5.5.3. The accuracy and performance of SystemClick is assessed with respect to the access models and with cycle-accurate RTL simulation of NOVA [208].

#### Accuracy and Sensitivity

SystemClick's accuracy compared to a real system and the sensitivity to changes of the architecture and of the memory mapping are vital for early yet quantitative design space exploration. We use the *headers* benchmark to expose effects of memory interference on system performance. It generates a burst of packets that is mainly processed by header-modifying elements of the WLAN TX path and then transfered to the PHY. By changing the mapping of packet headers from local to shared memory (cf. Sec. 6.5.1), memory effects can be exposed. Using a cycle-accurate SW profile, the simulated execution time of the single-PE baseline (C) can be matched almost exactly with the RTL reference.

Representing two typical exploration steps, the memory mapping – namely of packet headers – is varied and the application is partitioned onto a concurrent system comprising two PEs and a DMA. The consequences are different memory access profiles and thus total access latencies due to contention and memory distance.

- Benchmark (target) run times are summarized in Table 5.3. SystemClick's *accuracy* stays within 1 % of the reference if the architecture is changed, and within 3.6 % if the memory mapping is changed and significantly more shared accesses occur.

***Table 5.3*** *– Accuracy of memory access models when compared to cycle-accurate RTL simulation for two different architectures and memory mappings.*

| | | *Header mapped locally* | | *Header in shared memory* | |
|---|---|---|---|---|---|
| | | *Time* [ µs ]* | *Error* | *Time* [ µs ]* | *Error* |
| *Single PE* | RTL Simulation | 345 | – | 417 | – |
| | Single-Access** | 345 | **0.0 %** | 423 | **1.4 %** |
| | Grouped-Access*** | 345 | **0.1 %** | 429 | **2.7 %** |
| *Two PEs and DMA* | RTL Simulation | 239 | – | 306 | – |
| | Single-Access** | 241 | **0.8 %** | 309 | **1.2 %** |
| | Grouped-Access*** | 241 | **1.0 %** | 317 | **3.6 %** |

*) simulated target execution time for preprocessing a burst of 15 packets (*headers* benchmark)

**) SystemClick simulation with exponential distribution and ***) using the *grouped-avg* model

***Table 5.4*** *– Impact on system performance, i.e., target execution time as in Table 5.3, when migrating to a concurrent architecture or allocating more data to shared memory.*

| | *Changing Architecture* | | *Changing Memory Mapping* | |
|---|---|---|---|---|
| | *Headers Local* | *Headers Shared* | *1 PE* | *2PE+DMA* |
| RTL Simulation | -30.9 % | -26.7 % | 20.9 % | 28.1 % |
| Single-Access | -30.3 % | -27.8 % | 22.2 % | 28.6 % |
| Grouped-Access | -30.1 % | -26.1 % | 24.2 % | 31.3 % |

- Essential for exploration is the *sensitivity* of the models to such changes. From Table 5.4 it can be observed that both the performance improvement of migrating to the concurrent architecture (up to -31 %) as well as the penalty due to allocating more data to shared memory (up to 28 %) are reflected accurately in SystemClick. As expected, increasing concurrency is less benefitial for the mapping with more shared accesses due to memory contention.
- If *compared*, the single-access model outperforms grouped accesses marginally in terms of accuracy. This means that grouped accesses can well be used for design exploration, while single-access models allow verification of single design points.

### Simulation Performance

Simulation performance of an **architectural simulation** as in the preceding section depends on the chosen access model and the total number of shared memory accesses. Table 5.5 reveals the speedup over RTL simulation for the concurrent two-PE setup.[1] With headers mapped locally, the grouped-access model is about one order of magnitude faster than the single-access model and produces only 1/30 of the simulation events, resulting in a speedup of 50000X. Moving to the shared mapping, the tightly synchronized single-access model produces almost 40 % more simulation events reflecting the increased number of shared accesses. Performance is down by 22 %, while the grouped model remains almost unaffected. In fact, the grouped model performs only about 12 % worse than the simulation without any memory model. As a reference, we give the speedup of pure ISS (725X) and behavior-only simulation in SystemC (cf. Sec. 5.6.1).

[1] The results in [7] proved too pessimistic (2-4X) after revising SClick's settings and measurement metrics.

***Table 5.5*** *– Simulation speedup over cycle-accurate RTL simulation for* headers *with different memory models and system configurations.*

| | *Local Mapping of Headers* | | *Shared Mapping of Headers* | |
|---|---|---|---|---|
| | *Speedup [X]** | *SysC Events* | *Speedup [X]** | *SysC Events* |
| Two PE, single access | 6000 | 10740 k | 4700 | 14120 k |
| Two PE, grouped access | 50000 | 370 k | 47600 | 380 k |
| Two PE, no mem. model | 57000 | 365 k | 57000 | 365 k |
| Single PE, behavior | 286000 | 132 k | – | – |
| ISS | 725 | 132 k | – | – |
| X86 | > 2 M | 132 k | – | – |

*) over RTL with Modelsim 6.1, simulation time comprises 5 min. application initialization and 5 min. packet processing.

Previously [7], we have assessed the five-client setup of Section 5.5.4, which heavily stresses the **memory models**. In this case, the grouped-access model outperformed single access by up to 150X, mostly due to the 330X increase in SystemC events. Thus, the advantage of the grouped models can be one order of magnitude higher than given in Table 5.5. Similarly, judging from Table 5.1, we can expect architectural simulations for representative benchmarks to be in the range of up to 200000X faster than RTL (including a 10 % penalty for memory models) and reaching efficiencies between 2.5 and 10 %.

### 5.6.3 Discussion and Related Work

SystemClick enables both fast and accurate exploration and performance verification of protocol-processing applications. Trade-offs are available, e.g., fast grouped-access memory models can be used for design exploration, while more accurate single-access models allow verification of single design points. Simulation performance of single-PE models is approx. three orders of magnitude above ISS, with efficiencies relative to native execution averaging at 25 % and peaking at 74 %. Architectural simulations run 70X to 450X faster than a single ISS and up to five orders faster than RTL simulation. The accuracy remained within few percent, which we deem sufficient for early exploration, and was shown to reflect architectural changes properly.

SystemClick's **single-PE model** performs at least as good or better as existing work.[2] Interpretative ISS such as our MIPS reference runs approx. 4000X slower than native execution [161]. Both higher and lower values are found in literature, e.g., 16500X for Tensilica ISS by Cheung [42] or 1600X for Tricore ISS by Schnerr [214]. Modern compiled [198] or hybrid [64] ISS is reported only 68X resp. 83X slower. Revisiting the annotation approaches of Section 5.4.6, both Schnerr and Cheung emphasize aggregation of annotations prior to SystemC synchronization. They achieve a maximum efficiency of 6 resp. 3 %, one order of magnitude below our maximum efficiency. Notably, Cheung's accuracy remains only within 6 % despite fine-granular instrumentation. This could be due to neglected dynamic effects that can be captured better for larger blocks.

A closer look at the **benchmarks and simulator environments** is needed for better comparison. Unfortunately, no benchmark from our domain is widely accepted. Instead, multime-

[2] An exact comparison is difficult due to the lack of established metrics. In cases where no direct comparison to native execution is given, we use the frequency of the host processor as a reference instead, e.g., to derive the *efficiency* (simulated cycles to native cycles) or the corresponding slowdown.

dia or crypto benchmarks with very different characteristics are prevailing. For example, Schirner [212] reports an efficiency for SW-only simulation with function-level annotations of up to 48 % for JPEG. Such homogeneous kernels require very sparse instrumentation in SystemClick if the run-time data dependency parameter is used. Wang [240] reports up to 80 % efficiency for AES. But after embedding the simulator in SystemC, efficiency drops to 25 %. In fact, both approaches perform application scheduling natively, either implicit or using an RTOS model. While our results are at least comparable, it must be noted that SystemClick completely relies on the SystemC scheduler. This impairs simulation performance. However, SystemClick is therefore not limited to HW/SW co-simulation and rather allows to flexibly assign tasks to arbitrary resources for exploration (similar to the *virtual* models in [105]) and enables, e.g., high-level tracing.

From this perspective, we relate to architectural simulation and especially the **memory hierarchy**. Adding the memory facet significantly impairs performance of other approaches. Kempf reports a speedup of only 10X over LISATEK ISS [106] when memory accesses are issued directly to a TLM model. Even though not directly comparable, the performance of Wang decreases by at least one order of magnitude for relevant benchmarks if cache simulation is added. Our grouped-access model reflects penalties from shared resources at only 10 % performance degradation, outperforming single-access models by one to two orders of magnitude. This is comparable to the findings of Bobrek for their hybrid approach [29], who report 40X speedup over cycle-accurate accesses.

**Full system simulation** is hard to assess. RTL models of MP-SoCs are typically in the range of 100–1000 Hz, i.e., about 2 million times slower than native execution of the system function (in accordance with Tab. 5.5). Replacing processors with ISS instances improves this by 10-100X [200, 63]. If the architecture is represented with SystemC/TLM models, the speedups can be 100X up to 10000X [148]. The virtual prototype presented in [243] is in this range, running at 100 kHz. Surprisingly, the more abstract approach by Kogel [111] based on virtual architecture mapping comparable to SystemClick performs only marginally better. The abstracted approach of Kempf [105] achieves 100X over an ISS-based IXP co-simulator but neglects the memory hierarchy. In the range of SystemClick (single-digit efficiency, speedup over 50000X), we recognize the abstract TLM and function-level RTOS models of Schirner [212]. However, these are less flexible, run natively (as discussed above), and neglect at least parts of the memory hierarchy.

In **summary**, SystemClick's single-PE simulation is one order of magnitude better than fine-granular approaches and at least comparable to abstract or optimized frameworks, but is sufficiently accurate and flexible as needed for productive exploration. Adding the memory facet reduces performance by one order of magnitude in related approaches, whereas our models hardly degrade. Architectural simulation reaches four to five magnitudes above RTL, which is at least comparable to recent very abstract and restricted RTOS/TLM models while providing precise performance estimation of SW tasks and shared resource accesses. We conservatively estimate the speedup over full virtual prototypes to be in the range of two orders of magnitude, depending on the level of detail or the flexibility they offer. Considering the efficiency of our approach, significant speedups could only be expected if the SystemC kernel is further optimized or parallelized [58].

## 5.7 Chapter Conclusion

SystemClick with our extensions is a comprehensive development framework that supports all phases of our application-driven methodology: It aided application modeling in Chapter 3 and initial analysis in Chapter 4; it will abstract the performance of programmable platforms for early exploration of architectural trade-offs with fast turn-around times in Chapter 6; it will provide a path to implementation in Chapter 7. Having contributed to the initial implementation [9], we now summarize the extended framework and discuss its key characteristics.

We have **extended** SystemClick to counter the challenges posed by application complexity, real-time requirements, and the need for productivity when developing an efficient MAC system. Our graphical editor CliMMT [8] is library-based and together with the Click extensions of Section 3.2.2 provides a productive *application modeling* framework for MAC protocols. It is complemented by advanced *mapping* features for priority schedules and data structures that proved essential for exploring efficient application-to-processor and memory mappings. SystemClick's expressiveness and library of *architectural models* was extended for memories and shared communication, including, e.g., arbitrated FIFOs and cycle-accurate resource managers. Main contributions comprise:

- ***Automated performance feedback loop*** – A feedback loop integrating *profiling, simulation, and analysis* was introduced [4]. It enables early feasibility studies and proved essential for indicative platform exploration as well as verification and optimization of design points in terms of real-time constraints. The approach is centered around a single source instrumented at element boundaries and user-defined points. Precise performance feedback is obtained from ISS-based profiling that separates specifics of the target platform from the application. Profiling and simulation based on performance tags are available at different granularities and automation levels, up to the capturing of both computation and memory accesses. During simulation, performance is back-annotated from a database, orthogonalizing it from behavior and thus allowing quick evaluation of alternatives. SystemClick's infrastructure was optimized for simulation performance and for analysis based on tracing and statistics, especially improving turn-around during exploration.
- ***Memory exploration*** – The memory facet was added to SystemClick targeting heterogeneous hierarchies in deeply embedded systems [5, 7]. Access penalties from shared resources due to latency, arbitration, and synchronization can be reflected. Wrappers accumulate pre-categorized accesses to user-defined data structures and regions from profiled tags. Accesses for a given interval can then be reflected at two different levels of abstraction:
  - In *single-access* mode, cycle-accurate stimulus to abstracted memories or full SystemC models is generated. The distribution of accesses is chosen by the user or based on traces. This allows precise verification of single design points.
  - The *grouped-access* mode synchronizes groups of accesses by invoking iterative contention estimation functions in a hybrid simulation/analytical approach. Our algorithm yields upper penalty bounds and can be scaled to average contention. This mode is up to 100X faster than single accesses and thus well suited for identifying ideal memory mappings from hundreds of design points. Accuracy could be increased by at least 4X over a related approach.

The following **key aspects** make the extended framework especially suited for productive development and early exploration of efficient systems at the electronic system level:

- *Seamlessness* – Despite its light-weight nature SystemClick seamlessly covers our development process and design space: It comprises modeling, early verification of function and feasibility, automated performance feedback, high-level analysis, and code generation. All relevant facets including computation, HW/SW trade-offs, communication, and memory can be reflected (see Chap. 6 and [2, 5]). A single source of code is used, avoiding reimplementation and increasing productivity.
- *Appropriate abstractions* – The Y-Chart proved key to efficient development, as it allows to optimize application, architecture, and mapping separately [109]. SystemClick complements this with abstractions above the C language and RTL for:
  - *Application* – Click's modularity eases reasoning about application and mapping. User-defined instrumentation raises the abstraction to functional blocks that can be considered for optimization and that enable true high-level analysis. Platform-specific functions are explicit and abstracted as services.
  - *Mapping* – User-defined data structures and, e.g., packets are assigned to memory areas for mapping to storage resources. Click task chains remain explicit and can have, e.g., priorities. Performance is kept independent of behavior.
  - *Architecture* – Building blocks are abstracted and configured as needed (e.g., scheduling and performance). Tasks can be assigned flexibly to resources.

  This results in fast turnaround times during exploration (see Sec. 6.9). In addition, accumulating performance tags with accesses pre-categorized to memory areas is vital for efficient consideration of memory accesses in groups.
- *Quality of results* – The careful choice of abstractions and the flexible synchronization granularity ensure both high simulation performance for design exploration (as in Sec. 6.5.1) and accuracy for verification. Speedups reach up to five orders of magnitudes over RTL and at least two orders over virtual prototypes. Further improvements could only be achieved if SystemC is discarded for task scheduling, sacrificing mapping flexibility. Grouping memory accesses outperforms single accesses by up to 150X. The accuracy compared to the target system (RTL) remains within few percent and proved sufficiently sensitive to reflect architectural changes [7]. In general, SystemClick trades off speed and ease of use with accuracy. However, especially during early exploration accuracy can be misleading.
- *Extensibility* – The use of SystemC enables interfacing with existing IP, e.g., based on TLM. More complex modeling, e.g., for hierarchical communication and pipelined memory systems can be achieved from standard SystemC constructs and our extended library. SystemClick supports other domains and MoCs in principle. Last not least, the framework can be extended to the power facet, as the most relevant utilization data (memory, processor, communication) are known.

The framework's quantitative impact on **productivity** and the benefits of the application-driven methodology are discussed at the end of the respective chapters for each step. A summary is found at the end of the development cycle in the conclusion to this thesis.

# 6 Platform Exploration

This chapter investigates an efficient MAC architecture for state-of-the-art and future WLAN products. Such an architecture must be as programmable as possible to support evolving standards and to ease the development in short time-to-market cycles. Ideally, a platform is provided that can be targeted to different product configurations (e.g., access point and station) and that is adaptable to future protocol versions and related standards. At the same time, costs must be minimized: Chip area is the main cost driver in today's devices, and on-chip memory and processors are significant contributors.

In step four of our methodology – architecture exploration – we perform a search of the architectural design space based on the reference application. Since the emphasis is put on programmability, the search can be narrowed down to the exploration of a generic, modular, and programmable architecture – the NOVA platform.

During the process, we follow the Y-chart by optimizing the platform instance, mapping, and application separately. The starting point is a generic, single-core platform instance. This instance is utmost flexible but likely to fail other design objectives. The benchmark scenarios introduced in Section 3.4 allow the evaluation of architectural options during exploration of core type, multiprocessor mappings, the memory hierarchy, HW/SW trade-offs, and system communication. SystemClick together with accurate performance profiles is used to evaluate design points by simulation, capturing dynamic effects.

Eventually, a platform template optimized for MAC operation is presented that is customized and quantified for a range of WLAN device configurations. In addition, the productivity of our approach and its applicability to related protocols is assessed.

## 6.1 Design Space and Architectural Baseline

*Design space exploration (DSE)* is a well-researched topic in the field of embedded systems [246] and computing architectures [181] in general. Ample work is available on network processors [216] and their design space [70] (also in terms of their memory hierarchy [165]). In wireless systems, the focus is often put on the PHY layer (e.g., [127, 140]). However, little is found on flexible WLAN systems and their deeply embedded and real-time critical MACs: Other work is either too specialized, such as exploring multithreading [82] or instruction set extensions [91], or too generic [249]. Samadi [203] targets a specific architecture and RTOSes. Liu [141] focuses on a BlockACK implementation.

Instead, we need to explore IEEE 802.11n WLAN on a generic platform and for product-relevant device configurations. NOVA (cf. Sec. 5.1.1) is an ideal starting point: Its modularity exposes all relevant axes. It is not limited to specific processors or communication structures and can even be used to emulate other platforms. In addition, it is readily available and supported by SystemClick such that design points can be evaluated quickly. Leveraging

platform-based design, we limit the design space and customize, refine, and extend NOVA towards our *Wireless LAN on NOVA (wilaNOVA)* architecture.

### 6.1.1 Baseline Architecture Setup

The exploration starts with a single-core instance of NOVA. Figure 6.1 details its components: a single **processing element (PE)** is connected via **message passing** and a shared memory to generic host and IO interfaces.

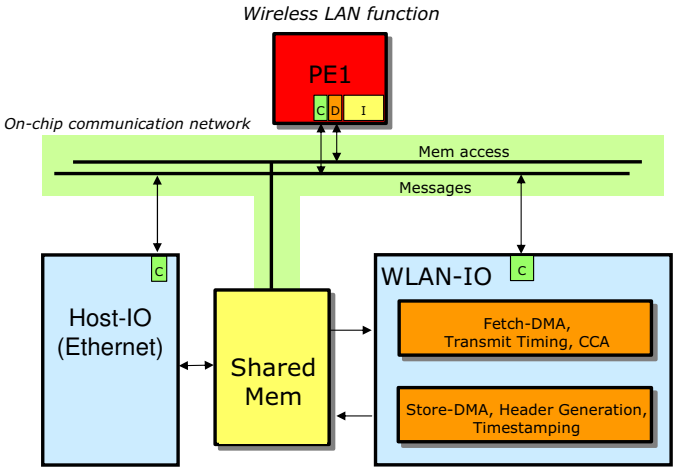



***Figure 6.1*** *– Basic single-PE instance of NOVA with generic host and WLAN IO interfaces.*

The two **IO modules** (Ethernet, WLAN) are decoupled by NOVA sockets. Consequently, all communication is based on packet descriptors (PDs) and/or system messages (SMGs). PDs are generated upon packet reception. The platform is optimized for packet processing and thus relies on DMAs to move packet data in and out of the system [207]. These data movers have a separated, direct access to the shared memory. Free memory segments are always available. The generic IO interfaces are a mere placeholder for concepts presented in this chapter (e.g., timely packet transmission and timestamping). The channel status (CCA) is sent to a PE via system messages. The *MIPS M4K* [160] is used as the PE's embedded **processing core**, for which the application is compiled with CRACC [210]. An application performance database has been profiled using instruction-accurate ISS (cf. Sec. 5.4). Ideal **(shared) memories** are assumed initially, i.e., memory accesses have no additional latency and are not subject to contention.

### 6.1.2 Design Objectives and Exploration Steps

Gries [67] divides the space for design exploration into an objective and problem space. Several of the **primary objectives** in [67] are relevant for WLAN:

- *Cost* – concerns primarily chip area. It is our main design driver and evaluated by summing up the areas of subcomponents. Memory is a major contributor, necessitating precise estimations. We deem development effort cost relevant as well. The architecture must thus be kept simple, homogeneous, and programmable.

- *Power* – is less important in wired systems. However, consumption has to be kept moderate, as fans or coolers increase costs. Low-power processes and power saving techniques thus are common. We do not explore the power facet but put reasonable performance bounds on dynamic resources such as processors.[1]
- *Speed* – has two aspects in WLAN: throughput – of the Preparation-Management (PM) layer, and real-time responses – in the Access-Response (AR) layer (see also Sec. 4.3.1). A configuration is only feasible if deadlines are met.
- *Flexibility and Scalability* – are meta objectives that cannot be evaluated in absolute numbers. However, we aim to keep the architecture as programmable and generic as possible, and rely on the NOVA platform that can be scaled as needed [208].

Our search thus is *multi objective*. It is aided by problem-specific **secondary objectives**. An important metric are resource utilizations of PEs, buses, memories, and buffers in SystemClick. Profiling results are available. HW-SW trade-offs are quantified in terms of speedup and area overhead. Specific to memories, clocking frequencies, stalls, and sizes are considered. Also, we evaluate our system in terms of reliability, deterministic behavior, and compatibility with future protocols.

A bottom-up coverage of the design space, i.e., the **problem space** along various axes is natural for platforms. Our benchmarks (cf. Sec. 3.4) drive the exploration and allow to examine the impact of, e.g., aggregation settings and PHY speeds. Leveraging SystemClick for quick evaluation of design points, the platform is explored along these axes:

- *Core type and software* – Two different cores are evaluated. Compiler settings and the Click overhead are considered for optimization.
- *Multiprocessor partitioning and scheduling* – Overcoming the performance limitation of a single core, the mapping is varied to achieve real-time performance and required throughputs. Alternative scheduling policies can increase efficiency.
- *Memory hierarchy* – Performance and cost considerations necessitate a heterogeneous memory architecture. We aim to mitigate the negative effects of latency and contention. In addition, memory requirements are re-evaluated.
- *HW/SW function split* – Optimizations of the processing element (PE) and an architecture for a *Specialized PE (SPE)* at the MAC/PHY interface are assessed. Protocol functions and platform services are optimized.
- *SoC communication and interfaces* – Message-based communication impacts the system, e.g., through back-log, and host interfacing must be evaluated.

## 6.2 Baseline Performance

The fully-programmable architecture establishes a baseline that guides further DSE steps. As dynamic effects in the application must be considered, the full system function is simulated in SystemClick. First, an evaluation setup is established, then performance requirements and the impact of protocol settings are assessed.

[1] A broad range of related work is available on power estimation for embedded systems [60], platforms [205], and at the system level [23]. Langen el al. [120] estimate power consumption in a network processor, and Lee and Mudge [126] address the problem for programmable WLAN systems in an SDR context.

***Table 6.1** – Functional performance categories.*

| | |
|---|---|
| **IO** | Overhead for packets and packet descriptors |
| **Operating System (OS)** | Incl. scheduling, task identification and starting, platform services |
| **Protocol** | All functions strictly related to the WLAN protocol |
| **Modeling** | e.g., paint annotations, switches, tokens, queues, schedulers |
| **Lookup and classification** | Classifiers and other lookup functions (e.g., for addresses) |

***Table 6.2** – Quantification of platform overhead (core instructions/cycles).*

| | |
|---|---|
| **PD transfer**. Packet descriptors (PDs) are moved from and to the message-passing FIFOs into the local memory by the core. We assume $2 + S_{PD}/4 * 3$ instructions with $S_{PD} = 64$. | 50 instructions |
| **Extra packet transfer latency**. In addition to the transport and arbitration delay on communication resources, extra latency is acquired, e.g., in the specialized PE. | 50 cycles |
| **Identify and start task chain**. For a received PD message the associated task chain must be identified from a dynamic list of candidates and must be started. | 12 + 7*$N_{IO-Tasks}$ insn. |
| **Schedule Timer**. The task must be sorted to a dynamic timer list according to its expiry. | 12 + 7*$N_{Act.-Tmrs}$ insn. |
| **Scheduling Loop**. Retrieve the next runnable task from the timer list and start it. | 26 instructions |
| **Memory management**. Allocation and freeing of memory blocks. | 40 instructions |
| **Standard packet functions and tokens**. All inexpensive cases profiled directly. | < 40 instructions |
| **Extended packet functions**. Join, split, and inexpensive clone (only PD affected). | 60 instructions |

### 6.2.1 Evaluation Setup

Evaluation of design points is performed by **SystemClick simulation** using an instruction-accurate performance database. Model run times representing up to 5 seconds proved sufficient to capture performance-relevant effects. Deterministic packet drop rates and rate selection are used for reproducible results.[2] The main **indication for performance** is the minimum frequency of the processing core(s) that is needed to execute the benchmark while fulfilling real-time requirements. The minimum is determined over several simulation runs. The frame context deadline (16 $\mu$s) is measured by *WifiConstraintChecker* elements (cf. Sec. 4.1.2). **Secondary indications** are resource utilizations and throughput. PE utilization (i.e., the relative time during which tasks are being executed) can be well below 100 % due to the real-time constraints. Instruction counts per element can be derived and are categorized into functional categories as in Table 6.1.

The platform architecture and its **performance overhead** are abstracted as summarized in Table 6.2. Components (PEs, IO modules) are represented by resource managers (RMs) as introduced in Section 5.2.4. Platform services (cf. Sec. 5.4.2) provide for communication, task handling, memory management, and packet operations. It is assumed that frequently used services are highly optimized, e.g., by coding in assembly language. The separation of application from platform concerns allows to establish an indicative baseline, while specific implementations can be explored later.

### 6.2.2 Performance Analysis

Application performance and critical paths are analyzed with focus on performance-relevant aspects. **Initial results** show minimum frequencies in excess of 70 GHz. Throughput is

[2]The packet drop rate is deterministic and set to 5 %. Within (PHY-layer) aggregates, 10 % of dropped packets have biterrors in the subframe header. Rate selection was performed but set statically to the maximum medium capacity for consistent results, i.e., 300 Mbit/s *PHY speed* for the 2x2 setup.

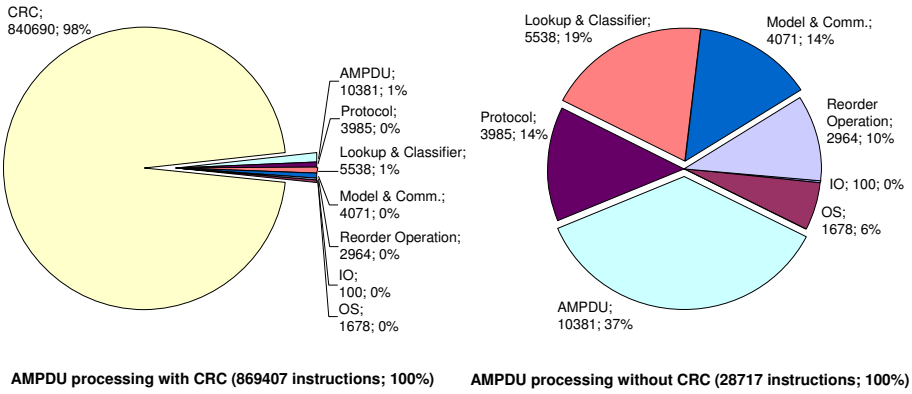



***Figure 6.2*** *– Instruction distribution for A-MPDU de-aggregation and subsequent BlockACK generation (16 subframes, TP scenario). CRC dominates (left) and thus is accelerated (right).*

uncritical, as PE utilizations are below 4 %. Instead, the critical constraint is the fitting of an incoming AMPDU frame and subsequent BlockACK generation into SIFS (De-AMPDU-to-BlockACK).[3] The TP scenario proved most demanding as it features the largest A-MPDU aggregation (16 subframes, up to 64 kB). A closer look at the functional distribution of the **critical path** in Figure 6.2 indicates that processing of complete A-MPDU packets within SIFS is not feasible (even if CRC is accelerated by HW):

- *De-AMPDU* comprising packet splitting, subframe header verification, and delimiter searching in case of errors accounts for 37 % of instructions in the TP benchmark. If all subframes failed or for 64 subframes, ten times more instructions are required. Thus, HW support for de-aggregation is needed such that subframes are forwarded one-by-one to the PE and only the last subframe must be processed in SIFS.
- *Model communication* and repeated redundant classification of each subframe account for 33 % of instructions and offer potential for optimization (cf. Chap. 7).
- *OS overhead* is moderate, and only 24% of the cycles are actually spent in *protocol processing* and reordering packets, leaving little to optimize.

For the **refined HW/SW splits** in Figure 6.3 we consider the data-flow operations CRC, cryptography, and (De-)AMPDU as hardware accelerators (as further discussed in Sec. 6.6.2). Mapping A-MPDU de-aggregation in HW alone reduces required performance by 75 %. CRC and crypto acceleration saves up to 97 %. The effects are almost orthogonal, and the combined solution needs only about 1 % of the initial frequency. Resource utilizations are slightly higher as well but still low in general. Peaking at 31 % in the transmitting AP of the TP scenario, an indication is given that pre- and post-processing can be significant contributors. The **critical path** is now dynamic A-MPDU aggregation after an incoming BlockACK (BlockACK-to-AMPDU). First, the BlockACK's bitmap is used to determine frames for replay, then the follow-up A-MPDU is generated within SIFS. Figure 6.4 details the functional distribution:

- *Replay processing* for incoming BlockACKs accounts for 40 % of instructions, but almost 33 % (model + lookup) are spent to identify and transport the frame to the correct Re-

[3] Performance required for response generation can be influenced by the internal application structure. Incoming frames are thus processed first in terms of the response generation and bursts are avoided by insertion of additional queues (see also Sec. 6.4). Channel access specifics are also considered later.

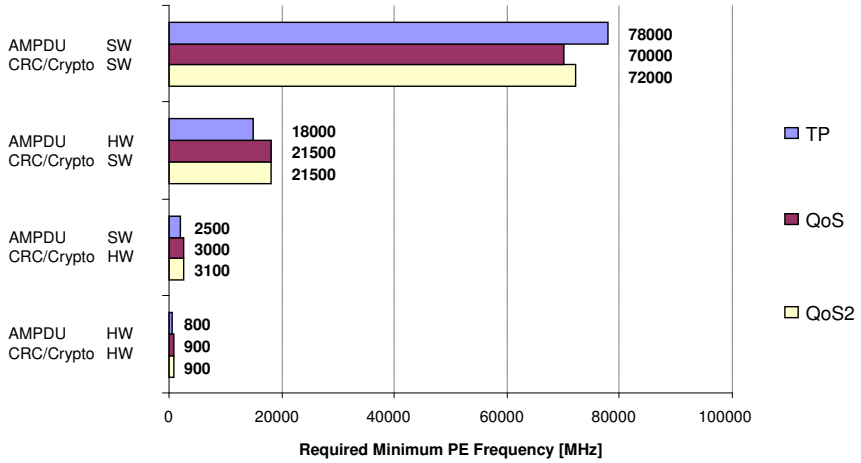



***Figure 6.3*** *– Comparison of HW/SW tradeoffs for aggregation and payload processing (CRC/crypto) for scenarios TP, QoS, QoS2 (less required frequency is better).*

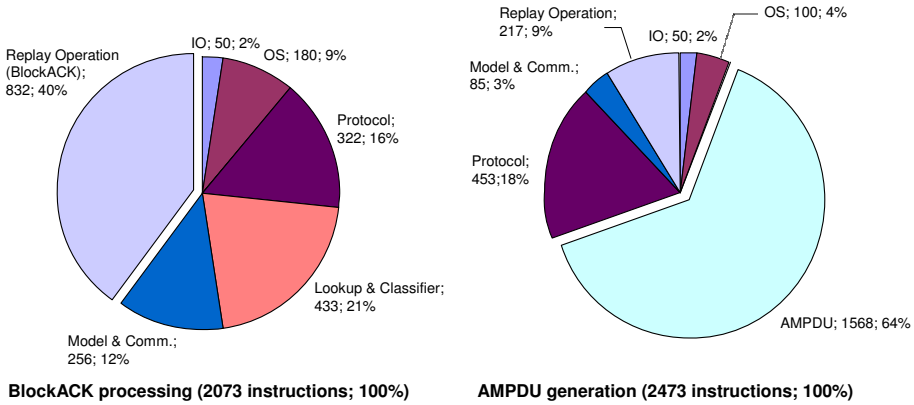



***Figure 6.4*** *– Functional distribution for processing an incoming BlockACK (left) and scheduling a follow-up A-MPDU with 16 subframes in the TP scenario (right) with HW support.*

orderBuffer. An optimization necessitates profound application changes as discussed during deployment (cf. Sec. 7.3.2).

- When an *A-MPDU* of size 16 is *generated*, 82 % of the instructions are already used in A-MPDU and protocol-specific operation, leaving little to be optimized.

A closer look reveals now that performance is impaired significantly by post-processing (mainly A-MSDU de-aggregation) that blocks the resource on receiving devices due to the platform's run-to-completion scheduling scheme. If the computational resource was always available upon packet reception, the PE would have to run at only 350 MHz to achieve the 16 $\mu$s deadline (BlockACK-to-AMPDU) in the TP setup. **Altogether**, improving resource allocation is the most promising step to increase resource utilization and to lower performance requirements. This is further discussed in Sec. 6.4.1. A-MPDU aggregates with 64 subframes remain critical but are unlikely to occur if A-MSDU aggregation is used, because only 16 packets of 4 kB match the 64 kB total aggregate limit. Such aggregates

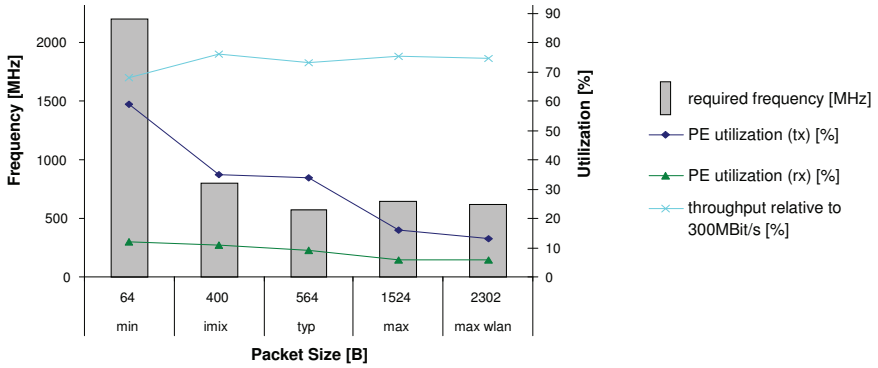



***Figure 6.5*** *– Required frequencies (bars), PE utilizations, and relative throughput for different packet sizes in the TP scenario (with A-MSDU and A-MPDU) at 300 Mbit/s PHY speed.*

should thus only be optimized as needed for a deployment, limited to less subframes, or generated statically (at the expense of reduced throughput).

### 6.2.3 Impact of Packet Size, Aggregation, and PHY Capacity

Figure 6.5 shows the impact of **packet sizes** (minimum 64 B, typical 564 B, maximum 1524 B, maximum wlan 2302 B) and distribution (IMIX [13] or constant) on system performance and throughput above the MAC layer in the TP scenario:

- Total *throughput* is between 204 and 228 Mbit/s, roughly 80 % of the capacity of the PHY layer (300 Mbit/s) due to overhead for channel access and PHY encapsulation. The maximum is reached in IMIX distribution, for which optimal fitting to TXOPs with large aggregates occurs. Increased protocol overhead for subframe encapsulation is limiting the throughput for small packets.
- *Performance requirements* are highest for small packets and IMIX due to A-MSDU deaggregation at the receiver (1.2 GHz suffice at the sender). Starting with typical-sized packets, real-time transmit processing becomes dominant and requires 600 MHz. Larger packets come with higher requirements, as A-MSDU aggregation is less efficient resulting in larger A-MPDUs (up to 27 subframes vs. 16 for IMIX).
- *PE utilizations* are highest for small packets due to pre- and postprocessing. Considering the computational power effectively used, the TP scenario with IMIX traffic could be run on one PE at 280 MHz at the sender if no real-time constraints were present. The load at the receiving station is approximately 2-5X smaller.

If aggregation at the MAC-layer (**A-MSDU**) is switched off, the impact of PHY-layer aggregation (**A-MPDU**) becomes apparent. Large A-MPDUs with up to 64 subframes and A-MPDUs restricted to 16 subframes are considered. Required frequencies range from 500 MHz for maximum-sized frames in restricted aggregates to 850 MHz for minimum-sized frames in 64-subframe aggregates. Restricting the aggregation size from 64 down to 16 can thus reduce requirements by 20 % for a given packet size. However, throughput is also reduced due to channel access overhead and PHY encapsulation, most notably for small packets (down to 36 %) and almost negligible for large packets. In addition, the **processing load** is higher if A-MSDU is switched off, as more MAC functions need to be performed per input

packet. Small packets require 77 % more cycles. For IMIX traffic, the penalty is still 63 %, but effectively drops below 10 % for larger packet sizes.

Finally, **PHY capacity** is considered. Since inter-frame gaps are unchanged and control frames are sent with basic rates, higher capacities do not result in a linear growth of throughput. Virtually no impact in terms of the required minimum frequency can be observed, as real-time response requirements remain unchanged. Doubling PHY speed to 600 Mbit/s increases the PE utilization to 60 % at the sender for IMIX traffic. Considering real-time constraints and computational complexity on a single-PE architecture, it can be projected that a throughput of approx. 800 Mbit/s would utilize the processing core at 900 MHz most efficiently. Only at 1200 Mbit/s, the minimum frequency has to be raised to 1 GHz at 95 % utilization for full packet throughput. The overall impact is moderate for relevant IMIX traffic and only critical for minimum-sized packets.

### 6.2.4 Summary

The simulation-based evaluation of our scenarios on the baseline architecture yielded a better understanding of the application's dynamic behavior and directs further exploration. Key to an indicative analysis are platform representation and the separation of platform services. We find in summary:

- A ***software-only*** approach on a single core is not feasible today. Consequently, expensive CRC and crypto functions must be put to dedicated HW. Integral for WLAN is A-MPDU de-aggregation in HW such that subframes can be processed one-by-one during reception and the PE is relieved from expensive error correction.
- With the ***refined HW/SW split*** all benchmarks run on a single 900 MHz PE. Critical is the generation of A-MPDUs following a BlockACK. The optimization potential here is limited and would require finetuning of the algorithm. However, given that the resource was always available, a 400 MHz core suffices for realistic setups.
- The ***performance impact*** of packet size and large aggregates can be considerable for very small packets, but other packet distributions remain feasible. Restricting A-MPDU size can lower requirements by 20 %. PHY speed has virtually no impact, as real-time requirements remain unchanged. The impact on other system resources such as memories must be examined further.

The availability of the resource for real-time processing is a major concern impacting performance requirements. Resource utilizations remain low (7 % to 30 %) and make different scheduling and resource allocation advisable (cf. Sec. 6.4).

## 6.3 Core Type and SW Optimizations

The anticipated choice of the PE's processor core is justified in the following by assessing performance/area trade-offs for WLAN. Further integral aspect of SW-dominated systems include compiler settings and the impact of the modeling language.

### 6.3.1 Processing Core

MIPS 32-bit cores [160] have proven a good performance-cost ratio in previous studies [210] and come with a mature toolchain. We focus on two cores: *M4K* [162], the smallest MIPS core, and *24k*, which is widely used in CPE products. Based on the **datasheet comparison** of Table 6.3a, the M4K can only be clocked at 400 MHz[4] but offers a better performance/power/MHz trade-off. Most importantly, the area footprint is significantly smaller. Many features of the 24k that increase its size are needless with the architectural assumptions of NOVA and for WLAN: Caches and complex fetch logic are not needed, as scratch-pad memories are preferred. Memory protection and paging is not needed by NOVA's thin OS layer. Arithmetic operations (MulDiv-Unit) do barely occur.

| *Feature* | *MIPS M4K* | *MIPS 24k* |
|---|---|---|
| Frequency* [MHz] | 200-414 | 750+ |
| Performance* [DMIPS/MHz] | 1.62 | 1.51 |
| Power* [mW/MHz] | 0.04 | 0.34 |
| Core Size* [$mm^2$] | 0.12 | 0.83 |
| Pipeline [stages] | 5 | 8 |
| Branch Prediction | no | yes |
| Built-in Caches | no | yes |
| Variable Page Size | no | yes |
| MulDiv-Unit removable | yes | no |

*) in 90nm, according to [162]

| *Architecture* | *CPI* |
|---|---|
| M4K Harvard architecture | 1.08 |
| M4K Harvard, no HW multiplier | 1.13 |
| M4K unified bus | 1.45 |
| 24k Harvard with caches | 1.35 - 1.4 |
| 24k Harvard, caches saturated | 1.3 - 1.35 |

***Table 6.3*** *– Comparison of datasheets and CPIs (WLAN benchmark) for MIPS M4K and 24k.*

A comparison of the cores' **performance/area ratio** for WLAN shows that the M4K is better suited for parallel and deeply-embedded systems: *Cycles-per-Instruction (CPI)* ratios were determined for our WLAN benchmark as shown in Table 6.3b. The M4K in Harvard configuration has a CPI of 1.08 but 1.45 for the unified memory interface. The impact of removing the multiply unit is small and limited to few elements (e.g., duration calculation). The CPI of the 24k is worse (1.35–1.4), even with saturated caches in the steady state (CPI 1.3 - 1.35). This is due to its additional pipeline stages and the complex fetch logic optimized for remote memory accesses. Altogether, the 24k offers 70 % more performance (CPI 1.35 at 750 MHz over CPI 1.13 at 400 MHz). However, four to seven M4k cores can be used instead of one 24k, depending on subsystem configuration.

### 6.3.2 Code Size / Performance Trade-offs

Compilers such as *gcc* have optimization settings that trade off code size with performance. Settings for speed (`-O2`, `-O3`) perform, e.g., loop unrolling, while `-Os` use heuristics to make code smaller [161]. Since both parameters may be critical, we have to find the best compromise for the WLAN application. Compared to the legacy application (11e), 11n proved more sensitive to changes. Our analysis shows:

- *Static comparison* of all 11n-related elements in the database (cf. Tab. 6.4) shows that code size can be reduced by 11 % resp. 23 % if `-Os` is used. The performance gain is

[4]Synthesis results for the MIPS M4K core show that a C65HP, rvt, 12-track library is able to meet timings at 400 MHz + 15% overconstraints on clocks.

***Table 6.4*** *– Static comparison of compiler settings for all instrumentation points in WLAN.*

| | *Code Size* | | | *Static Performance (total instructions)* | | |
|---|---|---|---|---|---|---|
| | *-Os* | *-O2* | *-O3* | *-Os* | *-O2* | *-O3* |
| | 44 004 B | 48 896 B | 54 084 B | 74 204 insn. | 72 282 insn. | 66 153 insn. |
| *-Os* | - | 11,1% | 22,9% | - | -2,6% | -10,8% |
| *-O2* | -10,0% | - | 10,6 % | 2,7% | - | -8,5 % |
| *-O3* | -18,6% | -9,6% | - | 12,2% | 9,3% | - |

***Table 6.5*** *– Run-time performance comparison for the QoS(11n) setup (critical path only).*

| *Optimization* | *-Os* | *-O2* | *-O3* |
|---|---|---|---|
| ***Required Frequency (MHz)*** | 355 | 325 | 300 |
| *-Os* | - | -8,5% | -15,5% |
| *-O2* | 9,2% | - | -7,7% |
| *-O3* | 18,3% | 8,3% | - |

somewhat smaller at only 7 % and 9 % if `-O3` is used. The `-O2` setting offers a good trade-off in terms of code size but its performance advantage over `-Os` is marginal.

- Most relevant for *run-time performance* is WLAN's critical path, as determined for *QoS* by simulation in Table 6.5. The gain from maximum performance optimization over `-Os` is now 18 % and the `-O2` setting offers a more reasonable trade-off.

In summary, the `-O2` setting is preferable as it offers the best trade-off. Both `-O3` with 23% increase in code size and `-Os` with an 18% performance penalty (when compared to each other) should only be chosen if clearly either run time or code size is an issue.

**Further optimization potential** arises from our detailed performance profiles at Click element granularity. In some cases, `-O3` does not produce the fastest and `-Os` not the smallest code.[5] Table 6.6 compares these settings with the absolute minimum over all options for all elements. The gain in code size is 3 % and some elements run up to 20 % faster. More importantly, a total potential of up to 26% in code size and 13 % in instructions is available for trade-offs: For example, in a multi-PE system there could be PEs running functions without timing constraints and PEs with critical functions. The mapping and optimization of elements can now be chosen to guarantee run-time performance under the constraint of optimal code size distribution. Such optimizations are often formulated as *Integer Linear Programming (ILP)* problems as in [227, 187].

***Table 6.6*** *– Potential gain from choosing compiler options per element and maximum trade-off possible if combining different options.*

| | *-Os* | *-O3* | *best per element* | *potential gain* | max. trade-off |
|---|---|---|---|---|---|
| *Code Size* | 44004 B | 54084 B | 42796 B | 2.7 % | 26.4 % |
| *Static Performance [insn.]* | 66153 | 74204 | 65835 | 0.0 %* | 12.7 % |

*) Single elements can perform up to 20 % better.

[5] Compilers trade off compiler run time with (partial) solutions of complex optimization problems. While reasonable compiler run times are required during development, the final product code could be optimized for several days, just like HW designs are. In addition, performance depends on the target architecture and can only be determined by profiling. This is, e.g., recognized in [16]: *Auto-Tuners* profile different variants [61] whereas others exhaustively search the optimization space [151].

***Table 6.7*** *– Runtime overhead of Click's modularity in the QoS2(11n) scenario.*

| | *Required Frequency* | Overhead |
|---|---|---|
| *Baseline Profile (Push/Pull Ovhd.)* | 375 MHz | – |
| *No Push/Pull Ovhd.* | 350 MHz | - 7 % |
| *Add. SimpleAction Ovhd.* | 425 MHz | + 13 % |

***Table 6.8*** *– Memory requirements for element state and graph connections can be reduced by 39 % by optimizing Click's memory overhead in the QoS(11n) application.*

| Add. Optimization | Element Memory | Port Memory | Total Memory | Saving |
|---|---|---|---|---|
| – | 18092 B | 5568 B | 23660 B | – |
| Slim element structure | 15044 B | 5568 B | 20612 B | 13 % |
| Combine pointers and ports | 13012 B | 2784 B | 15796 B | 33 % |
| Single connection | 13012 B | 1392 B | **14404 B** | **39 %** |
| (Element state /wo overhead) | 5854 B | - | 5854 B | 75 % |

Code compression is another means to reducing code size. True code compression is proposed, e.g., in [130]. In MIPS cores, code is not compressed but rather a simplified 16-bit instruction set is used. This instruction set comes with performance penalties and limited expressiveness [161]. Different approaches are compared in [129]. Since peak performance is critical for WLAN, code compression is not further investigated.

### 6.3.3 The Click Overhead

Modularity in Click entails overhead in run time and code size, as shown in Table 6.7. The overhead for push/pull indirection, i.e., the dynamic resolution of connections at run time is moderate (7 %) and reflected in our baseline. However, if agnostic (SimpleAction) elements are used that also resolve their type at runtime, the additional overhead is 13 %. Optimizations proposed in [210] include static resolution of agnostic elements that is stated to save up to up to 30 % depending on the application (13 % in our case). Push/pull overhead may be removed by connecting follow-up function pointers directly or by inlining. However, applicability is limited as elements such as switches rely on this indirection, and the code size will increase if elements are shared amongst tasks.

Memory overhead can be reduced as shown in Table 6.8. A slimmer element structure is achieved by removing function pointers. Port counts can be combined or statically compiled into most elements. In addition, arrays containing inport/outport (function) pointers and port numbers can be combined, if pointers are restricted to 3 B and ports to 1 B (at the expense of run-time overhead for masking). Storing both ends of a connection is not needed if the connection type is resolved prior to compilation. In total, the Click overhead for one element with one input/output can be reduced from 60 B to 28 B.

Since the element state without any overhead comprises only 5.9 kB, the theoretical memory cost for Click's modularity accounts for 59 % of the final figure of 14.4 kB with all optimizations applied. However, Click's elements and graph structure also determines control flow and data organization. This would have to be provisioned by an alternative all-in-one-element implementation likewise.

### 6.3.4 Summary

We have exemplarily compared a small micro-controller (M4K) with a standard embedded core featuring caches and a deeper pipeline (24k). The **MIPS M4K** was found better suited for our purpose, as it offers a better power/performance ratio and is 4–7X smaller, while the 24k offers only 70 % more performance in our architectural setup and for WLAN. The M4K can be clocked just above **400 MHz**, which means that feasible design points during further exploration are limited to this boundary.

Compiler settings and **SW optimizations** play an important role for SW-based WLAN systems. The *-O2* setting offers the best performance/code size trade-off and is thus used in the following. A per-element analysis showed that up to 25 % of additional trade-off potential is available if settings and mapping are optimized, e.g., as an ILP problem. Code compression is not applicable as peak performance is critical for our application. The **Click overhead** in memory can be reduced by up to 40 % from the existing implementation. From the final figure, still 59 % of memory are related to Click, but also include control flow and data organization. Runtime overhead can be removed except for push/pull function calls (7 %), which is already included in our profiles.

## 6.4 Multiprocessor Partitioning and Scheduling

The WLAN application has tasks that need to run immediately, such as for response generation. At the same time, long-running pre- and post-processing task may block the computational resource for extended periods. The extreme effect of this is shown in Figure 6.6 for an 802.11e setup implementing crypto functions in SW [9]. The response times for ACK and CTS frames in the single-PE setup are distributed over a wide range, both inside and outside the 16 $\mu$s frame context deadline. After partitioning the application and changing the allocation to computational resources, i.e., the mapping, all deadlines can be met with significantly lowered core frequencies.

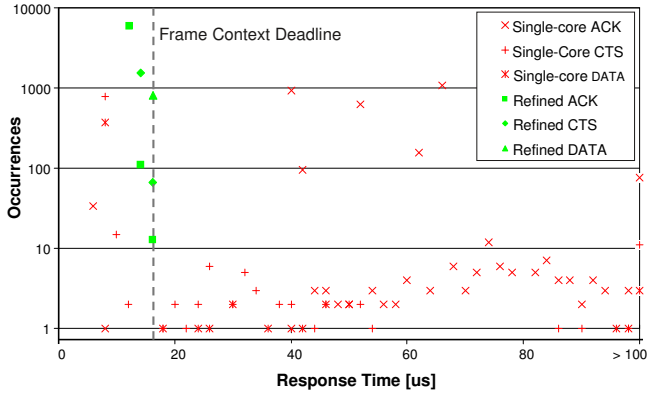



***Figure 6.6*** *– Histogram of response times for the 802.11e setup. After refining resource allocation the frame context deadline is met in all cases [9].*

A common solution to this problem is the introduction of generic **preemptive prioritized task scheduling**. However, this comes at the expense of a more complex operating system

and extra overhead for task switching and interrupt handling, putting the real-time behavior into jeopardy again unless expensive support in hardware is deployed. Thread switching and interrupt response times alone can easily exceed the cycle budgets in WLAN [220, 203]. In addition, many authors highlight inherent problems with interrupts and threads such as bad portability and their susceptibility to a variety of severe errors [195]. On shared-memory multiprocessors, concurrency correctness and the avoidance of race conditions and deadlocks are difficult to achieve [194]. Lee emphasizes that the thread model of computation has a lack of understandability, predictability, and determinism [124]. For example, a deadlock occurred in the Ptolemy project's kernel implementation after more than four years of intense code review. Thus, we aim to attack the problem within the Click model of computation.

Click has a **cooperative scheduling policy** [113], where tasks are started by external events such packet arrival and by internal timers. However, a task blocks the resource during its complete run time, and other tasks can only be started once the resource is available again. The standard scheduler in Click (and CRACC) is *First-Come-First-Serve*. As a consequence, the worst-case response time $t_{\text{response,FCFS}}$ for a critical task is

$$t_{\text{response,FCFS}} \leq \sum_{j \in \text{Tasks}} t_j.$$

SystemClick supports additional scheduling policies for each resource manager. For a priority schedule and assuming the critical path in question has the highest priority, the worst-case response time is still

$$t_{\text{response,prio}} \leq t_{\text{longest\_task}} + t_{\text{critical\_task}}. \tag{6.1}$$

Since our application is dominated by long-running tasks (e.g., stemming from bursts), a priority schedule alone has limited effect. In the following, two ways of **making** the **computational resource available** as early as possible for critical tasks are discussed:

- Exploring application-to-resource mappings, we *separate* critical from non-critical *tasks*. Additional task-level concurrency is exploited by pipelining and speculation.
- The reference application has been carefully optimized to defer processing in favor of immediate response generation. However, the process is manual and tedious. Instead, we systematically *decrease task lengths* and change the *scheduling*.

### 6.4.1 Multiprocessor Resource Allocation

The required frequencies, i.e., performance required for the single-PE mapping of Section 6.2.2 are not feasible in our target architecture's MIPS M4K core (limited to 400 MHz). Mapping the application to multiple resources allows to separate critical task of the Access-Response (AR) layer from non-critical tasks in the Preparation-Management (PM) layer (cf. Sec. 4.3.1). In addition, task-level concurrency within the layers can be exploited. This is examined in the following for the 2x2 access point (AP). SystemClick wrappers and ideal FIFOs abstract inter-PE communication (see also Sec. 6.7 and Tab. 6.2).

The results of **separating the AR and PM layers** onto two PEs are given in Table 6.9. Separated performance requirements for real-time processing and for data flow-like packet processing become apparent. The critical path at the AR layer requires just below 400 MHz

***Table 6.9*** *– Performance requirements at the AP for a two-PE mapping of the Access-Response and Preparation-Management layers (Agg. and crypto in HW).*

| *Scenario* | *Freq. (ARL)* | *Freq. (PML)* | *Limiting Constraint* | *PE Utilization at AP [%]* |
|---|---|---|---|---|
| scenario_TP (TX) | 600 MHz | - | BlockAck->Resp. | 44 |
| scenario_TP (TX) | 365 MHz | 250 MHz | BlockAck->Resp. | 5, 99 |
| scenario_TP (RX) | 800 MHz | - | Post-Processing | 8 |
| scenario_TP (RX) | 175 MHz | 50 MHz | AMPDU->BlockACK | 5, 99 |
| scenario_QoS | 900 MHz | - | Post-Processing | 7 |
| scenario_QoS | 325 MHz | 50 MHz | BlockAck->Resp. | 7, 84 |
| scenario_QoS2 | 900 MHz | - | Post-Processing | 8 |
| scenario_QoS2 | 375 MHz | 50 MHz | BlockACK->Resp. | 7, 87 |

***Table 6.10*** *– Performance requirements of different mappings for the TP(TX) scenario.*

| *Mapping* | *PE 1 [MHz]* | *PE 2 [MHz]* | *PE 3 [MHz]* | *PE 4 [MHz]* | *Throughput [Mbit/s]* | *PE Utilization at AP [%]* |
|---|---|---|---|---|---|---|
| 1 PE | 600 | - | - | - | 230 | 44 |
| 1 PE (static) | 500 | - | - | - | 180/210 (*) | 53 |
| 2 PE | 365 | 250 | - | - | 230 | 5, 99 |
| 4 PE (tx) | 365 | 85 | 85 | 85 | 230 | 5, 99, 99, 99 |
| 4 PE (static) | 165 | 50 | 165 | 70 | 180/210 (*) | 6, 18, 98, 96 |
| 4 PE (spec.) | 165 | 100 | 175 | 75 | >210 (**) | 6, 9, 99, 99 |

*) One aggregate per TXOP, ReplayBuffer with 32/64 packets **) Estimated, functional changes needed

and is related to A-MPDU generation after BlockACK. The long aggregates of the *TP* scenario and the complex scheduling of *QoS2* have the highest requirements. 175 MHz suffice for immediate BlockACK generation in receive-only mode. The requirements for the PM layer are determined by the actual throughput and range up to 250 MHz.

Further exploiting concurrency, the data flow-like **PM Layer** can be mapped to multiple PEs. It can be seen from Table 6.9 that the receive and transmit flows have diverging requirements and that such a separation is thus not promising. Instead, the PM layer is split into multiple pipelined stages as summarized in Table 6.10. Four PEs running at 365 MHz (AR layer) and 85 MHz (PM layer) are sufficient. The IO overhead of communicating the packets in between PEs is limited but will increase with more PEs.[6]

By contrast, the critical paths of the **AR layer** are highly sequential and cannot be parallelized at the task level as-is: First the state is updated, then aggregates are formed, scheduled, and transmitted. But application requirements can be relaxed at the expense of reduced throughput, namely dynamic length matching of aggregates and immediate retransmission of frames after BlockACK. Now the pull inputs of the EDCA that request new frames for transmission can be cut. Cutting with classic Click pull semantics, however, blocks the calling resource during the request and thus is of no avail. Instead, pulls must be non-blocking and produce packets ahead of time. Solutions include:

- *Static aggregation* of one fixed-sized aggregate per TXOP, pre-computed ahead of time or in parallel to channel reservation. This reduces throughput and requires a larger ReplayBuffer. As summarized in Table 6.10, *PE 1* requires only 165 MHz for EDCA functions and *PE 2* 50 MHz for A-MPDU processing.

[6] A detailed discussion of IO overhead on the NOVA platform is found in [207].

***Table 6.11*** *– Summary of performance requirements if AR and PM layers are separated.*

| *Layer* | *Feature/Throughput* | *Min. Freq.* | *Comment* |
|---|---|---|---|
| *Access-Response (AR)* | Dynamic TXOP continuation | 375 MHz | |
| | Static aggregation | 165 MHz | also for Lite-MAC |
| *Preparation-Mgmt. (PM)* | 120 Mbit/s | 120 MHz | 1x1, 150 Mbit/s at PHY |
| | 230 Mbit/s | 250 MHz | 2x2, 300 Mbit/s at PHY |
| | 450 Mbit/s | 490 MHz | 4x4, 600 Mbit/s at PHY |
| | 700 Mbit/s | 760 MHz | 1 Gbit/s at PHY |

- *Speculative aggregation* at fixed or a choice of sizes such that the best match can be selected. A higher throughput is reached at the expense of functional changes and increased code size (e.g., discarding of aggregates, replay buffer state consistency). The aggregating *PE 2* runs at 100 MHz, as it must be available at the second aggregation stage when follow-up subframes are transmitted back-to-back.

Table 6.11 sums up this section's **results per processing layer** as needed for further exploration. 375 MHz are required for the AR layer in standard configuration, which includes the functionality of a Lite-MAC. However, 165 MHz suffice for both scheduling and response generation if frame aggregation is relaxed. The PM layer requirements are scaled to throughputs as associated with different PHY configurations.

### 6.4.2 Task Partitioning and Scheduling

Separating critical tasks yields feasible solutions. However, utilizations of the real-time PEs remain low (cf. Tab. 6.10), and the question remains if a single-PE solution is feasible. From Equation 6.1 it is apparent that resource availability can be increased by reducing task chain lengths and by priorities, as is explored in the following.

#### Fully Partitioned Application Graph

A natural limit for task partitioning in Click is the element granularity. We call the systematic scheduling of a fully-partitioned graph with priorities ***Click Element Threading (CET)***, as introduced in Section 5.1.2. First, the flattened graph is extended with mapping information, then a script performs the graph transformation. All push task chains are cut into single elements by insertion of SystemClick wrappers and connected by FIFOs. The demand-driven pull chains starting at the EDCA are not cut, as stockpiling of packets may invalidate the ReplayBuffer. Then, **priorities** are now assigned in order to guarantee causality where necessary. Priority scheduling departs from Click's MoC, however, but is needed as cutting task chains can change application behavior. The outcome of this application-specific mapping process is shown in Figure 6.7. We note:

- The *pre-/post-processing tasks* of the PM layer are assigned low priorities, allowing for interruption. Elements producing packet bursts (e.g., A-MSDU) need to map their outputs to higher priorities such that they finish before the next burst.
- Higher priorities for *real-time tasks* in the AR layer must be chosen carefully because of semantic changes when all elements run independently. Especially the state-machines

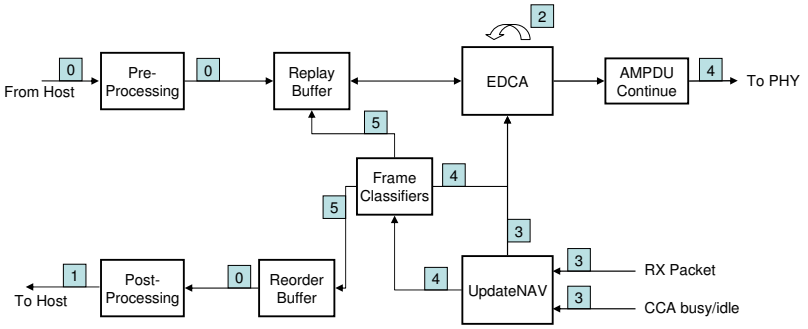



***Figure 6.7*** *– Application overview with priority mapping for a completely cut graph.*

(UpdateNAV, EDCA) rely on proper ordering of events. This is guaranteed by prioritizing primary over secondary outputs in Tee-like elements. In addition, EDCA timeouts must not interrupt real-time tasks.

### Priority Scheduling in NOVA and SystemClick

SystemClick wrappers representing graph cuts can easily be configured to priority scheduling for exploration. Of course, this is different for NOVA, where PEs must be extended to run both IO and timed tasks with priorities. It must be noted that a single scheduling queue in NOVA is insufficient because expired timers must overtake low priority timers if scheduling is delayed due to run-to-completion. If a single message passing (MP) inbox is used for multiple priorities, tasks must be rescheduled in SW. Then, reading all messages from a single inbox avoids head-of-line blocking at the expense of variable scheduling delays and lack of MP back-pressure. A small number of MP inboxes may be considered statically in the scheduler, e.g., low/high priority, at reduced expressiveness.

An implementation supporting arbitrary, programmable priorities for both timers and IO while maintaining MP back-pressure is sketched in Listing 6.8. An array of memory-mapped queues is checked for entries, returning the entry itself for a timer queue (in SW) or a non-zero entry for an MP inbox (in HW). Timers can be run right away. For an IO task, the packet descriptor is read out of the MP queue and the task is identified and run. Considering performance, only a single queue has to be checked for each priority, since a given priority is either assigned to IO or timers. For a complete dequeue, all queues need to be checked starting with the highest priority until a runnable task chain is found. Consequently, we add a scheduling cost on top of IO and timer activation as follows:

$$\mathrm{c}_{\mathrm{priosched}}(\mathrm{task}) = cyc_{fixed} + (N_{\mathrm{prios}} - prio_{\mathrm{task}}) \times cyc_{iter}$$

### Click Element Threading

Several implementations for priority-based re-scheduling of elements on the same resource (**CET scheduling**) are possible in NOVA. The insertion of queues in the application is too expensive as all queues need to be checked at a given rate. Instead, the NOVA scheduler and message passing (MP) is used: (1) Leaving the platform unchanged, a packet descriptor (PD)

```
while (true)
    for j = N_prio downto 1
        tmp = check_queue_head(j)          // returns task (timer) or true (IO)
        if (tmp)                           // queue was non-empty
            if (j is IO)
                pd = read_from_inbox(j)    // queue j is MP inbox
                t = identify_task(pd)
                run_task(t, pd)
            else
                run_task(tmp)              // tmp is a timer task
            break
```

***Figure 6.8*** – *Algorithm to schedule and run message IO and timers with priorities in NOVA.*

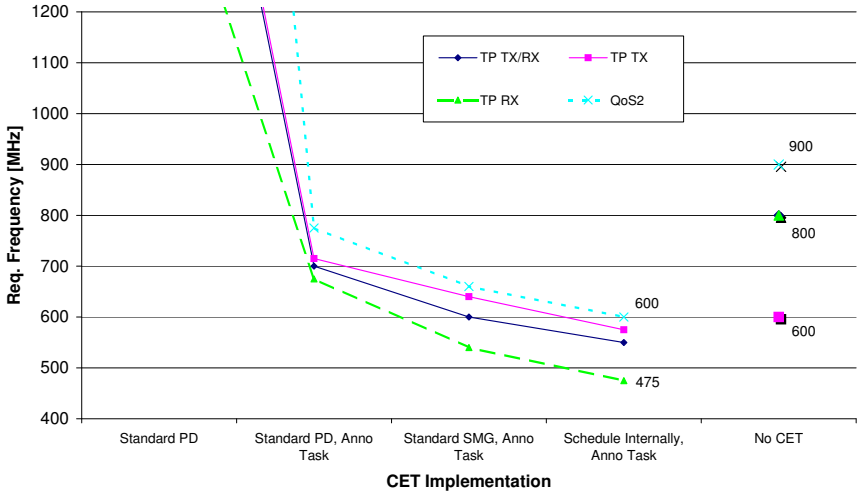



***Figure 6.9*** – *Performance impact of different implementations for* Click Element Threading.

can be sent to the same PE via MP. This incurs overhead due to IO and task re-identification. (2) The latter is avoided by annotating the task pointer to the PD context. (3) IO overhead is reduced by sending a shorter system message (SMG) and keeping the PD local. (4) All approaches relying on IO may result in deadlock. Consequently, extending NOVA for scheduling the tasks directly is the best solution.

The implementations are assessed in Figure 6.9. Leaving the platform unchanged (1) is not feasible. Option (2) is already better than a single-PE without CET (cf. Sec. 6.4.1). Reducing IO from (2) to (4) lowers requirements by another 20 %. In total, requirements decrease by almost 40 % for the QoS2 and TP/RX scenario due to better resource availability with CET. As a drawback, transmitting at maximum throughput (TP/TX) benefits only slightly (550 MHz at 87 % utilization vs. 600 MHz at 44 %) as the resource is highly utilized and scheduling overhead (up to 45 %) impacts the critical path. This overhead could be reduced if critical tasks are not partitioned and small elements are grouped. Regarding memory, elements producing packet bursts increase requirements if the contexts needed for scheduling are larger than the state kept internally, e.g., for de-aggregation. These elements should instead reschedule themselves every few subframes.

Even though single-PE performance is not in the range required for 2x2 setups, **CET** improves resource availability and utilization and has the following benefits for WLAN:

***Table 6.12*** *– Instruction budgets for background tasks in a single-PE mapping.*

| *Frequency [MHz]* | *Early Offset* [ µs ]* | *Total Budget (SIFS) [insn]* | *Budget for Background Task** [insn]* |
|---|---|---|---|
| 600 | – | 8000 | 3000 |
| 400 | – | 5333 | 333 |
| 400 | 4 | 5333 | 1667 |
| 400 | 16 | 5333 | 5667 |

*) requires application-aware scheduling **) assuming 5000 instructions in the critical path

- Devices with low throughput (1x1) or less functions (Lite-MAC, highly-optimized 2x2 AP) are feasible on a *single PE* with CET.
- In a *multi-PE system*, CET can be used to increase resource utilization, e.g., by mapping background tasks on real-time PEs. The total budget for non-critical tasks fitting inside the 16 $\mu$s frame context deadline is shown in Table 6.12.

The cycle budget for non-critical tasks could be further increased by **application-aware scheduling**: If it can be predicted when a resource will be needed by a critical task, it can be blocked for other tasks ahead of time. Such indications include the CCA or an estimation of the packet's end. An implementation could either use a polling element to keep the resource busy or be more tightly integrated with the scheduler. Table 6.12 reviews the budget for background tasks, when response generation is protected by an offset. The earlier the resource can be blocked, the longer other tasks may run without interfering with critical functions. Despite this promise, additional HW support (e.g., channel access, Go-Signal) and considerable application changes (events requiring immediate responses) are needed. Thus, application-aware scheduling is not further explored for now.

### 6.4.3 Summary

The main reason for high performance requirements and low resource utilizations are low-priority tasks of the PM layer that interfere with real-time processing in the AR layer due to Click's run-to-completion MoC. Two solutions were investigated:

- ***Separating layers*** in a multi-PE mapping exposed requirements just below 400 MHz for the AR layer and 75–250 MHz for the PM layer of the 2x2 Access Point. Higher PHY speeds do not affect real-time requirements but PM processing scales linearly with throughput. Pipelining is better than separating TX/RX parts. If aggregation is restricted, e.g., for Lite-MACs, less than 200 MHz suffice for real-time processing.
- A single-PE mapping with ***reduced task lengths and priority scheduling*** (i.e., Click Element Threading) was explored, extending NOVA and Click. Real-time constraints can be met at 400 MHz in principal, but throughput requirements for 2x2 or faster systems are too high due to scheduling overhead. If carefully considered, though, it can be applied to configurations with low throughput or reduced functionality, or in general to improve resource availability and utilization.

The exploration is continued with the focus put on the two-PE and multi-PE mappings, as these are more generic and scalable. Single-PE solutions require optimizations and changes to application and platform, but are in reach of a redesign if more effort is spent. The potential cost advantages are thus also evaluated in Section 6.8.

## 6.5 Heterogeneous Memory Hierarchy

Memories can be dominant cost factors in SoCs [178]. State-of-the-art WLAN MACs are often dominated by on-chip memory (or memory interfaces). Consequently, memory usage and memory organization must be a first-class concern, but so far an ideal shared memory architecture with unlimited capacity was assumed.

A typical **performance/area trade-off** is between local and shared on-chip memories. On the one hand, local memories can be accessed quickly and are always available but are dedicated and cost extra. Shared memories, on the other hand, are flexible in their usage and require (for the same total size) less area. However, they introduce access penalties that are difficult to estimate due to the effects of latency, arbitration, and clock synchronization. The memory models of Section 5.5 are used to reflect these penalties.

It must be ensured in the WLAN MAC that (1) the required throughput is reached and (2) real-time constraints are met. At the same time, chip area must be minimized and previously identified performance requirements (on number and frequency of cores) should be maintained. As the system function is very complex and characteristics of the application's layers are clearly distinct (cf. Sec. 4.3.2), exploration can be facilitated by considering effects impacting throughput (AR layer) and full system real-time performance (PM layer) separately. On this basis, we optimize properties of the system architecture, the mappings of data memories, total memory usage, and memory management.

### 6.5.1 Optimizing Memory Hierarchy and Mapping for Throughput

Properties of the system architecture that impact system **throughput** (such as memory distance and the mapping of application data) are optimized in the following while keeping the on-chip memory footprint as small as possible. A throughput above the MAC layer of 450 Mbit/s is chosen as the exploration target, as is needed for a high-end 4x4 WLAN AP. The system function is mapped to two PEs (PM and AR layer) running at 200 MHz. Including three DMAs for packet IO and an encryption engine, five clients access a single shared memory.[7] The memory access system is a crossbar running at memory speed with timing as in Figure 5.10 on page 81: A request delay of one PE cycle occurs for shared memory accesses, with an additional response delay of one cycle for reads. Arbitration takes one memory cycle and additional read memory latency cycles reflect the distance of the shared memory. A *close* on-chip memory is represented by no extra latency, whereas more *distant* or off-chip SRAM requires three extra cycles. See also Section 7.3.3 for a discussion of memory latency in the prototype implementation.

#### Impact of the Memory Subsystem

Mapping all data to shared memory, we observe from Figure 6.10 that none of the configurations achieves the full WLAN throughput. The performance bottleneck are the PM-layer functions being executed on the first PE. Performance is down 30 % due to request and arbitration delays, even when PE and a close memory run at the same speed. Moving to

[7] This setup was published as a case study in [5]. The TX path of the WLAN reference is adapted and real-time constraints are removed by abstracting the EDCA function. The performance values given in the following are indicative for the reference scenarios or slightly better due to these simplifications.

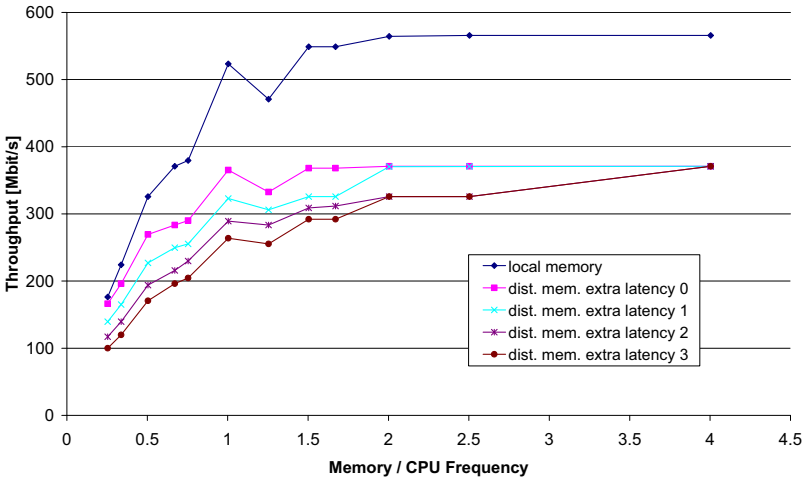



***Figure 6.10*** *– Increasing shared memory distance and the effect of access contention lowers system performance, shown over varying memory clock speeds.*

distant memory, the performance drops again by 30 %. Increasing the latency further (e.g., to represent SDRAM) starves the PEs significantly and is not appropriate for frequently accessed data structures in uncached systems. In addition, the effect of synchronizing the local PE clock with the memory subsystem also decreases performance if no proper fractions are chosen. E.g., at 1.25X the PE speed, the penalty is 12 % not accounting for additional synchronizing logic that would be required. We thus limit the exploration to memories running 0.25, 0.5 and 1X the PE speed.

### Impact of Memory Mappings

Mapping parts of the application's 200 kB data memory to closely-coupled private memories mitigates the adverse impact of shared memory latency and contention. But such **local memories** are expensive, need to be allocated for each PE, and limit the flexible use of the platform's resources. Thus, we aim to keep these as small as possible by considering six data **memory areas** to be mapped either to locally or shared. These areas include stack, local heap and element state, program code data (ELF), and packet descriptors (see also Sec. 4.2.2). A packet memory and global configuration need always be shared. Representing an optimization step of the application itself, we also map packet headers locally as part of the packet descriptor (at the expense of increased SW complexity and a small memory penalty).

The result of an exhaustive search of all 64 mapping combinations is shown in Figure 6.11, with **Pareto-optimal data points** achieving 450 Mbit/s being highlighted by large symbols. For a close shared memory (cf. Fig. 6.10, extra latency 0) running at 200 MHz the target can be met with 12 kB of local memory for element configuration, ports, and stack. Up to 31 kB of local memory are needed for a shared memory at 100 MHz. However, this can be reduced to 13 kB if the packet header also kept locally. A distant/off-chip shared memory (3 cycles extra latency) must run at twice the speed (200 MHz) for the same mapping. At the

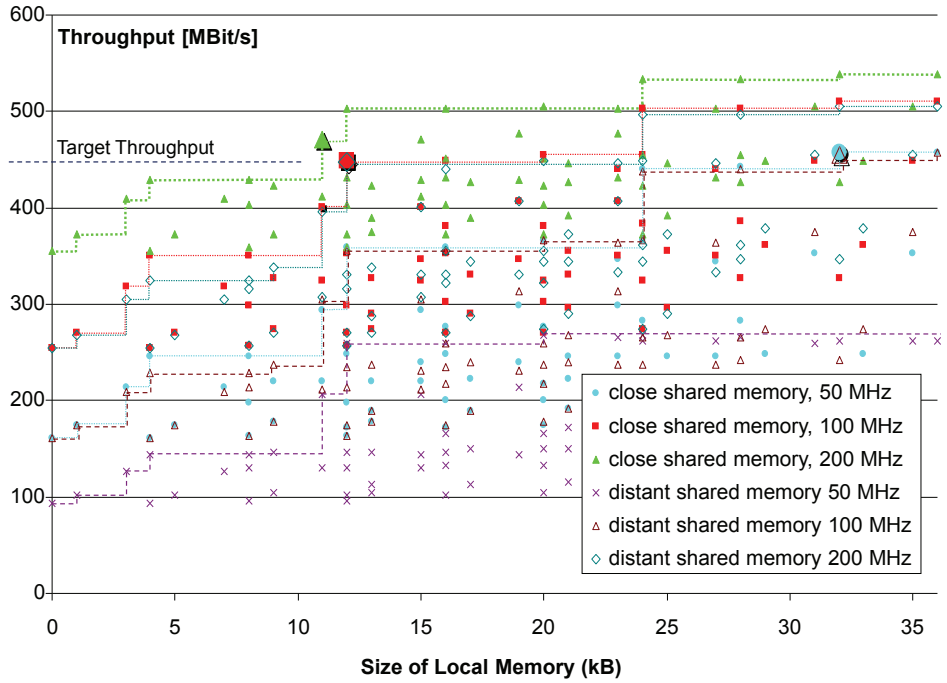



***Figure 6.11*** *– Impact of memory mappings reflected by local memory size (less is better) on system throughput (more is better). Pareto fronts and data points achieving 450 Mbit/s target throughput are highlighted.*

expense of 32 kB, when only queues are shared, a close memory even at 50 MHz[8] and the 100 MHz distant memory become feasible. The **penalty** of moving from a close/on-chip to a distant/off-chip memory at the same speed is between 28 to 42 % if all data are shared, and between 6 to 23 % if all possible data are mapped locally.

### Impact of Packet Flow and Contention

As five clients access a single shared memory, contention leads to limited performance. The DMAs and the WEP crypto accelerator account for most of memory traffic, whereas the PEs working on packet headers and shared data are the performance bottleneck. It can thus be advisable to allocate a *separated packet memory*. Furthermore, the coupling of the accelerator is inefficient: By transferring the entire packet payload, the memory traffic is effectively doubled w.r.t. normal IO. This increases the required bandwidth and access contention. An alternative is to group operations on the packets' payload (such as encryption) at the end of the task graph. Now, these operations may be performed *flow-through* close to the MAC/PHY interface, which accesses the payload anyway.

The resulting improvements are summarized in Figure 6.12. For a setup where all memory is shared, performance is improved by almost 12 % for a slow local memory and up to 5 % for other setups, as contention is reduced significantly. Distant memories are dominated by latency and not affected as much. If all possible data are mapped locally, the gain is less

[8] At 50 MHz the memory system's capacity limits a sustained throughput.

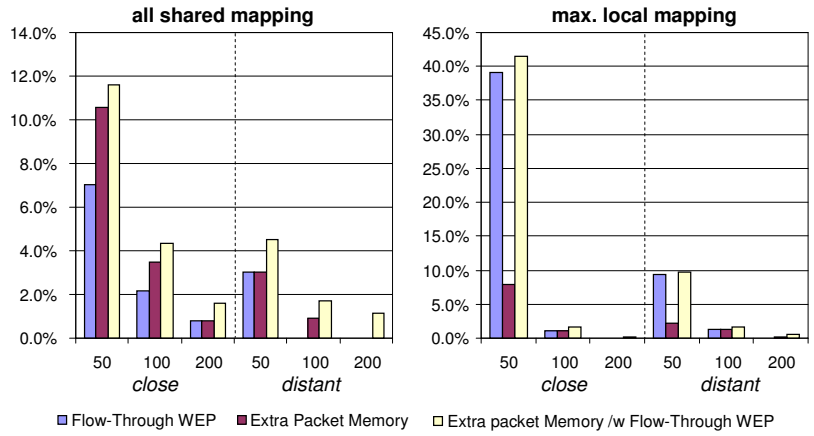



***Figure 6.12*** *– Performance improvements by making the WEP accelerator flow-through at the IO interface and by allocating a separated packet memory.*

significant. However, the throughput with a 50 MHz shared memory, which was previously limited by memory bandwidth, benefits greatly if the accelerator is made flow through. This is especially important for low-end systems with slow (memory) interfaces or very high throughputs, where an optimized data flow is essential.

### Discussion

The memory subsystem can severely impair system throughput. However, the effect can be limited if clocking, memory architecture, and mappings are chosen appropriately:

- The memory subsystem's frequency should be a proper divider of the PEs' frequencies to avoid *synchronization effects* (the performance penalty reaches 12 % otherwise). Higher clocked memory subsystems yield little benefit, also due to PE request delay.
- *Allocating* only 5 % (or approx. 12 kB) of total *memory locally* improves performance by up to 25 % and allows to reach the target throughput for a close shared memory running at 200 MHz. For a shared memory at 100 MHz, 15 % of the data or the packet header optimization is needed (then 6 %). While mapping of stack and element state locally is vital, the exact mapping of other data depends on the full application and is refined in the following sections.
- *Distant off-chip memory* for frequently accessed shared data should only be used if it is a design requirement. It comes at the cost of decreased performance (up to 42 %) and in general requires more local memory (e.g., 12 to 31 kB for 100 MHz memories) and higher memory frequencies (e.g., more than 200 MHz instead of 100 MHz).
- The total *impact of contention* on throughput was up to 20 % in some cases but dropped, e.g., to < 3 % with large local memories at 100 MHz. It is advisable to optimize the architecture with separate packet memory and flow-through accelerators, especially if the bandwidth is critical at very high throughputs or for slow memories.

Thus, our further search is focusing on close on-chip memories due to their performance advantage, but off-chip memories can be traded off if needed. The memory frequency is set

to 0.25x of the fastest PE clock to avoid synchronization effects, e.g., 100 MHz with 400 MHz PEs. An extra memory distance of one memory cycle is assumed, as is a realistic choice for shared on-chip memory in the given frequency range (cf. Sec. 7.3.3).

### 6.5.2 Real-time Performance with Heterogeneous Memories

The performance of the full WLAN application mapped to a two-PE configuration with a heterogeneous memory hierarchy is discussed on the basis of the previous section. After re-assessing packet storage implications at the system level, it becomes apparent that accesses to shared data such as PDs and queues are becoming critical for **real-time performance** at the AR layer. From a baseline, mappings are optimized to minimize local memory usage with respect to application features.

#### Packet Storage and System-level Implications

Consideration of packet storage for the full system is important, as the requirements for low-latency on-chip packet memory would be significant (e.g, up to 1 MB for the 2x2 AP). Fortunately, the analysis in Section 4.3.2 and Table 4.6 has shown that accessing a packet's payload is barely necessary. In fact, **protocol processing** on the PEs can be reduced completely to *packet descriptor* or *header* processing, if:

- *Packet headers* are buffered locally, e.g., in the PD. 32 B suffice for WLAN.
- *Payload processing* functions are moved to the end/beginning of the Click task chains, i.e., to HW accelerators close to the MAC/PHY (cf. Sec. 6.5.1).
- Appropriate *logical packet operations*, e.g., for concatenation and insertion of data segments exist (e.g., based on linked lists in the PD).

Since no more direct accesses are needed, **packet storage** does not impact real-time performance. Payload can be stored remotely, e.g., on the host processor or external RAM. The flow of packet data can be separated to minimize interference with on-chip buses by connecting the IO module directly to the host interface. The complete packet thus is only available immediately prior to transmission or after reception in between the host and PHY interfaces. All expensive payload-related operations (e.g., encryption) must be performed in this *payload processing path* (cf. Sec. 6.7.2).

Only minor **limitations** exist: Even if remote accesses are possible, these should not be time critical and be performed concurrently. For example, payload access is needed for A-MSDU de-aggregation to parse subframe boundaries. This can be done remotely as it is not time critical, or with the help of a filter unit in the payload path (cf. Sec. 6.6.2). Few management frames (such as Beacons) require payload exceeding the packet header, but such frames are uncritical as well. Uncompressed BlockACK control frames, however, require the analysis of 128 B (the more common compressed version fits in the header). A solution is redirection of these frames to on-chip memory by a filter unit. Or, if not time critical, they can be returned or processed by the host.

***Table 6.13*** *– Real-time performance and* additional *local memory for different mappings.*

| *Memory Mapping* | *Scenario QoS2** | *Scenario TP*** | *Add. Local Mem.**** |
|---|---|---|---|
| *Ideal memories* | 375 MHz | 365 MHz | – |
| *Only global data shared (queues local)* | 395 MHz | 375 MHz | 20.5 kB |
| *Global and queues shared* ***(baseline mapping)*** | **410 MHz** | **400 MHz** | **16.4 kB** |
| *Global, queues, and PDs (/w headers) shared* | 525 MHz | 525 MHz | 0 kB |

*) maximum A-MPDU size 8 subframes **) 16 subframes
***) mapped to local instead of shared memory, in *addition* to standard structures

### Real-time Performance Baseline

As a **baseline mapping** for full-system evaluation we allocate element state, stack, heap, ELF, PDs, and headers locally (as in Section 6.5.1). Global data structures, queues, and enqueued PDs are accessed in a close on-chip shared memory, as summarized in Table 4.6 on Page 56. Most shared accesses occur at the PM layer and their throughput impact has been discussed. The real-time **critical paths** in the **AR layer**, however, access shared data as well: The sending station's internal ID is classified for incoming packets, and for aggregates the initial packet is dequeued and total length is determined based on available packets in the queues. Few elements check configuration from the SIB.

Real-time performance requirements are up 10 % for both TP and QoS2 scenarios with the baseline mapping (shown in Tab. 6.13). Mapping queues locally as well is better for the TP scenario, where larger aggregates need to be pre-calculated by accessing queue memory. At the same time, classification and thus SIB accesses have a smaller effect in the TP scenario, as fewer stations are associated. An *additional* 16.4 kB resp. 20.5 kB are to be moved from shared to *local memory* for queues and PDs ((on top of standard structures, as estimated in Sec. 4.2.2). However, if these are also shared for minimum local memory, more than 500 MHz are required for both scenarios.

### Optimizing Mapping, Application, and Architecture

The baseline mapping of Table 6.13 both requires a large local memory and violates our performance limit (400 MHz). In addition, it does not scale well: larger A-MPDUs impair performance (460 MHz for 64 subframes)[9] and supporting more aggregation streams increases the additional local memory requirements (32.8 kB for 32 streams, 4x4 device).

Hence, our exploration must optimize mapping, application, and architecture in order to better trade off performance, local memory, and scaling of application features (aggregation). The following measures are considered:

- *Lookups* for incoming packets impair real-time performance due to shared accesses by approx. 4 %. Especially for a large number of connected stations, it is advisable to buffer the lookup structure locally (e.g., 512 B for 2x2 AP) or to add an accelerator. Buffering requires house keeping but can be used flexibly as needed.
- Local packet descriptors and headers speed up processing by up to 20 % at the expense of more local memory (16 kB for 256 packets). However, only the PD of the first packet

[9] Only the penalty from additional shared accesses is considered neglecting the computational impact. The quality of this discussion is still valid, however, independent of the current implementation.

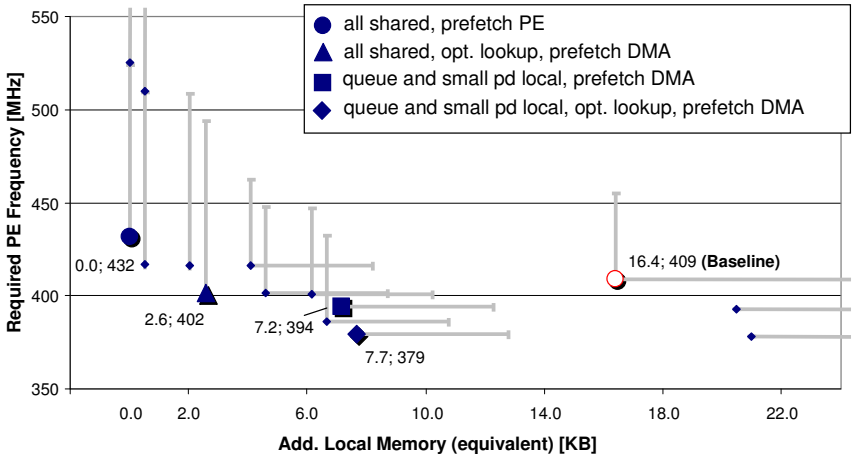



***Figure 6.13*** *– Trading off implementations w.r.t. required PE frequency (less is better) and additional local memory (less is better). Bars indicate performance penalties for A-MPDUs with 64 subframes (instead of 8) and memory increase when doubling the number of aggregation streams to 32.*

of an aggregate (the *boiler plate*) is needed for critical response processing. Thus, *prefetching* this PD only improves performance without additional memory.

- Prefetching can also be done by a *DMA*, e.g. in parallel to aggregation scheduling. The performance gain is moderate (approx. 4 %) and we estimate an equivalent to 2 kB of local memory as implementation cost.

- Both queue memory and PDs must be accessed once per subframe during A-MPDU generation. Memory latency thus is more critical for larger aggregates. However, as only the packet length is needed from the PD, a *small placeholder PD* of 4 B holding only the packet length and a pointer to the full PD in shared memory suffices.

The requirements for design points implementing these measures are summarized in Figure 6.13. The best design point that keeps all data shared (and thus needs no additional memory even with more agg. streams supported) and that uses the PE to prefetch the first PD requires 430 MHz. However, maximum-sized aggregates can only be scheduled at more than 500 MHz. Optimizing lookup and using a DMA, 400 MHz are almost sufficient but now 2.6 kB are required and large aggregates remain critical. Other design points that keep aggregation queues and small PDs local are a good compromise in that they allow large aggregates and face only a moderate increase in local memory with more streams. Combined with a DMA, 7.2 kB memory and a performance just below 400 MHz suffice for the 2x2 access point. Adding lookup optimizations, the impact of memory latency can almost be mitigated completely in the **recommended setup** at the expense of just below **8 kB** additional equivalent memory (diamond shape in figure).

Depending on the choice of implemented measures, aggregation size and number of streams can be traded off flexibly if either memory or performance are becoming critical. In any case, a tiny DMA function for PEs is advisable. For example, the AMPDU_Continue function (see Sec. 3.3.4) also benefits: It processes follow-up PDs in aggregates, and PD prefetching can account for 28 % of the cycle budget alone.

***Table 6.14*** *– Code size comparison (-O2) for single-PE and two-PE mappings.*

| | *Single-PE* | *PE 1 (**PM Layer**)* | *PE 2 (**AR Layer**)* | *Total (Overhead in %)* |
|---|---|---|---|---|
| *Init-only Code [B]** | 13156 | 7532 | 8736 | 16268 (+24%) |
| ***Runtime Code [B]*** | 60896 | **35864**** | **37260** | 73124 (+25%) |

*) init code can be overwritten after initialization **) incl. 12 kB placeholder for mgmt. functions

### 6.5.3 Memory Distribution and Organization

Multi-PE mappings and additional local memories increase the memory footprint. Furthermore, the amount of memory that can be accessed directly by a core in one clock cycle is limited by complex addressing and memory structures. The exact footprint must thus be re-calculated for code and data (total requirements are found in Sec. 4.2). In addition to static memory, dynamic usage for packets and descriptors is assessed.

The distribution of **code memory** to local memories is revisited in Table 6.14 for the two-PE setup. As the functions are split, less than 40 kB per PE suffice for the run-time code including management functions. Overall code size, however, has increased by almost 25 % over the single-PE mapping due to replication of OS and standard Click functions. For a Lite-MAC, the AR-layer code suffices. Code memory may be **insufficient**, e.g., if more management functions are needed. In this case shared memory can be used as overflow storage for uncritical code and data. A DMA function may relieve the PE from explicit fetching. Another option is the sharing of local memories between PEs, which increases flexibility in memory assignment without increasing the overall area footprint. However, there may be performance penalties due to concurrent accesses. For a stand-alone AP that requires several MB of code, caches can be used in addition to local memories but come with area penalties.

The distribution of **static data memory** is revisited in Table 6.15, again for the two-PE setup. Management and configuration memory is completely allocated to shared memory. Mapping options for other data were discussed in Section 6.5.2. The setup that keeps queues and all PDs shared requires 7/13 kB of local and 57 kB of shared data memory, but fails target performance. Mapping queues and PDs to local memory, however, requires 35 kB. The **recommended setup** uses a tiny DMA function and holds small PDs, lookup structures, and aggregation queues locally, totaling at only 20 kB with stable performance requirements. The total overhead due to local replication of data is below 2 %. AR-layer requirements (max. shared) suffice for **Lite-MACs**, where aggregation is performed on the host. Thus, significantly less memory is needed for queues, PDs, and management, totaling 35 kB and 21 kB for AP and station, respectively.

The application requires **dynamically allocated memory** at run-time only for packet payload segments and descriptors (and few uncritical management tasks). Static start-up heap requirements and the number of packets and PDs were discussed in Section 4.2.2. Our platform thus must provide a platform service (cf. Sec. 6.6.3) for management of shared memories that keeps allocation latencies for inbound traffic at a minimum:

- *PDs* are received by NOVA PEs via message passing and stored locally. As their size is fixed, allocation can be simplified. If PDs are swapped out to shared memory for permanent storage, e.g., in queues, a global memory manager is needed. As their total number is limited, short 2 B pointers suffice. In addition, memory compression could be applied to FIFO queues [196].

***Table 6.15*** *– Distribution of functional data memory (2x2 setup, all in [B]).*

| *Mapping* | *Req. Perf.* [MHz] | *Mem.* [B] | **Access Point (AP)** *PM Layer* | *AR Layer* | *Total* | **Station (STA)** *PM Layer* | *AR Layer* | *Total* |
|---|---|---|---|---|---|---|---|---|
| Max. shared | 525/431* | Global | 56760 | | 56760 | 16424 | | 16424 |
| Setup | | Local | 7446 | 14320 | 21766 | 4877 | 7050 | 11927 |
| Queues and | 375 | Global | 36280 | | 36280 | 11304 | | 11304 |
| agg. PDs local | | Local | 7446 | 34800 | 42246 | 4877 | 12170 | 17047 |
| **Recommended** | **379*** | Global | **52664** | | 52664 | 15400 | | 15400 |
| **Setup** | | Local | **7446** | **19952** | 27398 | 4877 | 8394 | 13271 |
| Lite-MAC | (200) | Global | 21332 | | 21332 | 9727 | | 9727 |
| | | Local | – | 14320 | 14320 | – | 11927 | 11927 |

*) requires tiny DMA function for prefetch

- *Packet payload* is moved directly to and from external (host) memory. Thus, additional management structures are needed for exchanging allocated and freed segments and to keep track of split segments (as needed for A-MSDU aggregation). Segments should be pre-buffered to reduce latencies. As a consequence of indirect packet operations, packets must be defined as arbitrary lists of memory segments that may, in parts, be allocated dynamically in on-chip memory.

### 6.5.4 Summary

This section discussed the configuration and architecture of the memory subsystem and the impact of mapping on throughput and real-time requirements. Local memories are to be kept at a minimum as they add cost, decrease flexibility, and may limit core performance. A feasible solution for the 2x2 access point[10] was identified:

- The memory subsystem's severe impact on ***system throughput*** can be mitigated almost completely, if clocking, memory architecture, and memory mapping are chosen carefully: Synchronization effects must be minimized by running the system synchronously. Allocating a small amount of memory (element state, stack, current packet headers and descriptors) locally yields a performance gain of at least 25 %. Close shared on-chip memories are preferable for frequently accessed shared data.
- Keeping packet payload on-chip is expensive, and packet moving increased the impact of shared-memory contention to up to 20 %. At the same time, ***protocol and payload processing*** can be separated: Protocol processing only requires access to a packet's header and descriptor, if indirect packet manipulation is provided. Payload processing can be grouped and allocated to dedicated HW close to the MAC/PHY interface, where the payload is available anyway. Thus, the flow of packet data can be decoupled completely (see also Sec. 6.6.1 and Sec. 6.7.2).
- ***Real-time performance*** is critical if aggregation queues and PDs as needed in the critical path are mapped to shared memory. Local allocation increases memory requirements significantly. Thus, it is recommended to maintain a small placeholder PD for packets in aggregation queues locally and to prefetch the next required PD. Together with a tiny DMA function and optimized address lookups, only 8 kB of additional local memory are needed and 380 MHz suffice for the AR layer.

[10] The findings are extrapolated to other device configurations for the cost analysis in Section 6.8.3.

- A distributed system with two PEs necessitates reconsideration of ***memory distribution and management***. Code memory requirements are up 26 % due to the replication of common functions and OS, but less than 40 kB per PE suffice for run-time code. The chosen mapping for data maximizes shared over local memory. Only 8/20 kB are needed locally, and requirements increase only moderately for longer A-MPDUs. 53 kB are allocated to shared memory, which must be partly managed dynamically (as further discussed in Sec. 6.6.1).

## 6.6 Hard- and Software Extensions

Selected hardware accelerators for computationally expensive and payload-related functions in WLAN offer an enormous performance gain over software implementations, as discussed in the baseline analysis of Section 6.2.2. Furthermore, the separation of header and payload processing and remote storage of packet payload is essential for achieving cost, throughput, and real-time targets (cf. Sec. 6.5.2). This puts the WLAN IO module into focus, as all payload-related functions must be grouped at the MAC/PHY interface, where the full packet is available. In the interest of flexibility and scalability – potentially to other protocols –, (1) only few accelerators should be used (2) that are as generic as possible and (3) that can be productively integrated to the system.

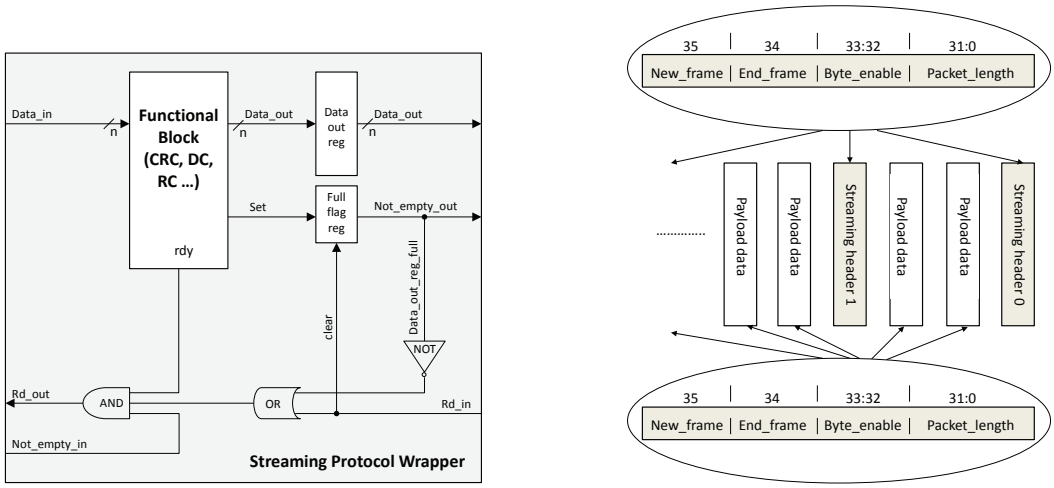
In this chapter, essential hard- and software extensions for our *wilaNOVA* architecture template are devised. The WLAN IO module is extended by generic shared-medium MAC functions and a modular payload processing path that makes it easy to map functions of Click elements into HW accelerators. This path then accommodates the small set of accelerators required for WLAN in our approach. Finally, the need for indirect packet operations due to remote packet storage motivates us to revisit platform services.

### 6.6.1 SPE - a Modular MAC/PHY Interface

The IO module surrounding the MAC/PHY interface is an integral part of *wilaNOVA*, as it autonomously transfers packets and encapsulates interface-specifics and payload-processing functions. It is based on NOVA's generic IO module and reuses message passing, DMAs, and configuration mechanisms. This module is now extended to a ***Specialized PE (SPE)*** optimized for shared-medium MACs. This includes improved packet handling to better accommodate the **separation of packet header and payload**, such as embedded packet headers and sequences of PDs with payload lists.

The scope of interface-specific functions required for shared-medium protocols with complex PHY layers exceeds basic forwarding and receiving of service data. Thus, such **generic shared-medium MAC functions** concerning channel feedback, timing, and PHY-layer control are added to the SPE module template:

- *Channel feedback and robustness* – Feedback to the PEs is provided via system messages, indicating channel state changes, collisions, and successful transmissions. Robust handling of PHY errors is required.
- *Timing* – A time stamp is available system-wide, in PDs, and to precisely schedule transmissions at the interface. A Go-Signal sent to the SPE is deferred to the annotated time and then initiates the transmission, while packet data can be provided later.




**(a)** FIFO-like streaming interface. **(b)** Streaming protocol

***Figure 6.14*** *– The modular streaming protocol put in place in the SPE's payload processing path. The packet-based interface extends Click's modularity onto hardware blocks.*

Packets must be discarded and the PEs be notified if this deferral is interrupted by a received packet.

- *PHY control* – Configuration interfaces are provided. Internal signals indicate started, completed, and ongoing transfers and can trigger special configuration sequences. In-band RX contexts are parsed into PDs.

The SPE furthermore provides a systematic approach to accommodate flow-through HW accelerator blocks operating on complete packets and their payload. Such blocks are grouped together close to the MAC/PHY interface since the payload is available only here due to the separation of header and payload processing and remote packet storage. This **modular flow-through payload processing path** available in receive and transmit directions comprises two aspects: Blocks are connected through a *FIFO-like interface* as shown in Figure 6.14a. It is based on the signals `not_empty_in` (indicating available data from the source) and `Rd_in` (indicating free space at the destination) and is able to operate continuously in pipelined mode. Additional registers are only needed if data are to be added or removed from the stream. The abstraction in between blocks is the packet, which is achieved by a *streaming protocol* as shown in Figure 6.14b. This protocol is established by a streaming header that moves in-band with packet data. It comprises, e.g., packet length and processing annotations. Extra side-band information indicates start and end of packets as well as byte enables, resulting in a data path width of 36 bit. Compactor modules adjust byte enable information.

This processing path effectively extends NOVA's modularity and Click's packet abstraction onto flow-trough HW blocks, allowing to integrate functions of Click elements easily. Blocks may even be generated from the Click description using high-level synthesis, as the (packet) interface is well defined. All blocks can be configured either on a per-packet basis via in-band annotations or directly through system messages. Blocks rely on the streaming protocol to modify packets (changing packet length), remove or add packets (changing new and end frame signals), and to communicate with downstream blocks (using annotations).

Additional FIFO buffers can be inserted to increase the cycle budget in case PEs are involved in direct configuration support, e.g., for the crypto engine.

### 6.6.2 Dedicated Hardware Accelerator Blocks

#### Required Accelerators

Only few dedicated hardware accelerators are needed in our wilaNOVA approach. At the system level, the SPE must be complemented by **packet movers** at the host interface to check interface queues for available packets, to generate PDs, and to hand off packets (cf. Sec. 6.7.2). Inside the PE's subsystem, a general-purpose **tiny DMA** is needed to locally access MP queues and for swapping data to/from shared memory. This includes storing and fetching PDs, code, management data structures, and updating headers to and from the packet's payload. The DMA is accessed through registers containing command and result queues for independent, non-blocking operation. Transactions are identified by an ID number. In steady state, the processor can initiate a transfer (write) every time the result of the previous one (read) is consumed.

All **other needed blocks** relate to flow-through processing at the MAC/PHY interface and are integrated into the SPE's modular path. Such blocks are commonly found in communication interfaces [207] and can also be adjusted to other protocols:

- *Payload processors* are needed for CRC and cryptography. En-/decryption blocks for WEP require *rc4*, whereas WPA/WPA2 is based on *AES*. Such blocks can significantly contribute to chip area, depending on their throughput [251]. Key tables and configuration are needed on a per-packet or per-flow basis. For this, direct access for PEs must be available, e.g., via the SPE's configuration infrastructure.
- *Enhanced packet functions* are required for A-MPDU (de-)aggregation. A block concatenates aggregated packets (TX). Subframe detection, error recovery, and logical splitting of packets are required for de-aggregation (RX). The streaming protocol is re-established such that the modifications are transparent for other blocks.

#### Optional Accelerators

Packet functions can be generalized into **parser** blocks inside the SPE's receive and transmit path. Implementations can range from simple hard-wired classifiers to ASIPs and can be loosely supported by a PE. As a minimum, a micro-coded *Scan-and-Edit* engine must be able to forward, remove, edit, and insert data into the payload stream. Such parsers can be used for filtering, classification, header translation, redirection of frames to on-chip memory, or assist in A-MSDU de-aggregation.

**Further accelerators** can be added as needed. Blocks for response generation (e.g., CTS/ACK) as commonly found in traditional HW-centric WLAN systems are not needed in our approach. In fact, they are discouraged, as functions related to the EDCA include complex state machines and protocol specifics that are difficult to implement and verify in HW and that especially limit flexibility. Better options include:

- The *NOVA PE's* local subsystem can be extended by closely-coupled accelerators to aid the operating system in terms of task scheduling and local memory allocation. A global wall clock timer register simplifies system timing. The MIPS instruction set can be extended [162], e.g, to efficiently access short 2 B-pointers or for bit manipulation as is commonly needed in protocol processing (see also [207]).
- *A-MPDU aggregation* is handled in SW for flexibility, but A-MPDU aggregate length could be efficiently determined by HW to optimize the critical path.
- *Channel Access* is aided by the SPE's timer, but the Go-Signal must be generated as fast as possible for response generation (see Sec. 4.1.1). Fortunately, most required data can be gathered statically ahead of time such that only few header fields and the EDCA state need to be considered[11]. It can be generated by an optimized Click element (see Sec. 7.3.2) or by a programmable HW classifier.

### 6.6.3 Exploration of Platform Services

Platform services concern OS functionality and parts of the packet library (cf Sec. 5.4.2). Their abstraction so far proved sound and enabled fast exploration, while keeping concerns separated. Since they can have a big performance impact, implementation options for two services – memory management and packet concatenation – are now exemplarily discussed, which are also required and representative for indirect packet operations.

#### Memory Management

The current approach to management of dynamic memory in NOVA suffers from **drawbacks** that must be reconsidered in the light of real-time applications such as WLAN:

- The *current implementation of* `malloc/free` [59] was chosen for its small footprint (< 1 kB). Average performance and worst-case response time, however, are insufficient when the memory is fragmented or compaction of freed blocks is needed (see Tab. 6.16). Other algorithms such as [153, 150] perform better but entail too much memory overhead. Instead, careful optimizations include supporting only fixed segment sizes and running compaction as a background task. Buffering of free segments reduces latency at the cost of wasting a small part of the usable memory.
- Presently, a *centralized memory manager (MM)* handles requests via message passing. While the performance impact is hidden by splitting such transactions, response times dependent on the availability of the MM's PE and of the message system. As a careful optimization, lists of fixed-size segments can be pre-allocated in shared memory. More elaborate would be the use of multiple, distributed MMs. These MMs can (1) be restricted to a distinct memory part, e.g., chosen statically or at runtime via messages. Or (2), multiple MMs can manage memory simultaneously if management structures are in shared memory and properly synchronized.

While we apply said careful optimizations for prototypical deployment in Chapter 7, a **dedicated memory allocation HW block** is advisable that is directly accessible through registers. HW-supported methods are described, e.g., in [218] or in [103]. The latter propose to use combinations of fixed size segments to utilize memory more efficiently.

[11] Only with TXOP continuation the Go-Signal depends on BlockACK and replay buffer state in rare cases [10]

***Table 6.16*** *– Exemplary performance of NOVA's memory manager during the transmission/reception of 200 packets (including initial allocations).*

| *Memory Management* | *Total Calls* | *Average Instructions* | *Worse-Case Instructions* |
|---|---|---|---|
| malloc | 7194 | 320 | 9852 |
| free | 6122 | 40 | - |

#### Packet Operations

***Table 6.17*** *– Pre-processing performance and cost trade-offs for* `packet_append`*.*

| *Implementation* | *Throughput** | *Improvement* | *Cost*** |
|---|---|---|---|
| Re-allocate and copy payload | 221 Mbit/s | | 24.5 kB |
| Re-allocate and copy with DMA | 428 Mbit/s | 94 % | 24.5 kB, DMA |
| Linked list of PDs | 445 Mbit/s | 101 % | 24.5 kB + 86 kB |
| Linked list of payload segments | 435 Mbit/s | 97 % | 24.5 kB + 13 kB |

*) In the situation of the case study of Sec. 6.5.1 **) Memory for PDs as discussed in Sec. 4.2.2

Two types of packet operations are needed for WLAN. *Basic* operations such as `clone` and `kill` need to guarantee a low latency. This is discussed during implementation (see Sec. 7.1.1). *Extended* operations for indirect packet modification including `split`, `append`, and `insert`, however, have a complex impact on the system's throughput and its memory requirements. Consequently, we discuss **Memory/performance trade-offs** for the `packet_append` operation as required for A-MSDU aggregation in the following: A straight-forward implementation re-allocates the first packet to a sufficiently large segment and copies all other payloads. Even assuming only one re-allocation per aggregate, it can be seen from Table 6.17 that performance is poor unless a DMA is used for data moving. But in both cases there is increased data traffic in the system and copying may be impossible if payloads are stored remotely. Copying can be avoided by indirect concatenation. Accumulating PDs in linked lists, however, requires up to 86 kB of additional memory (as in Sec. 4.2.2) and incurs additional costs for list traversal.

*wilaNOVA* thus organizes packet payloads indirectly in pointer lists referencing segments in shared or remote memory. The first pointer is part of the PD, and additional pointers are stored in linked list descriptors. This results in high performance at a moderate memory overhead (13 kB) for list PDs. More importantly, organization in linked segment lists enables to perform **all packet operations** in a similar **indirect manner** by modification of PDs and locally available segments only: Packets and their headers and payloads can be arbitrarily modified prior to passing to the host or SPE given that scatter-gather operation is supported by these interfaces.

### 6.6.4 Summary

This section investigated platform extensions in hard- and software towards the MAC-optimized, flexible, and efficient *wilaNOVA* architecture template:

- The WLAN IO module is extended to a ***Specialized PE (SPE)*** by adding generic shared-medium MAC/PHY interface functions and a modular payload processing path. Modules in this path are connected through a FIFO-like interface and streaming protocol.

This establishes a packet abstraction in between blocks that effectively extends the modularity of Click onto such HW accelerators.

- Only ***few accelerators*** are needed in our approach in addition to data movers at the system level. These packet payload processors (CRC, crypto, A-MPDU) can be integrated productively into the SPE's processing path. No additional, dedicated hardware is required for response generation, but the PE itself and WLAN channel access can be further improved if necessary.
- Revisiting ***indirect packet operations*** it was shown that lists of packet segment pointers in packet descriptors are an efficient implementation option. Modification then only requires scatter-gather capabilities at the interfaces. For memory management, a dedicated block is advisable if careful optimizations are not sufficient.

## 6.7 System Communication

Communication is an important aspect of MACs. It is needed both internally for accessing IO modules and other PEs and externally for interacting with the host processor when integrated, e.g., in a residential gateway. The NOVA platform provides message passing and autonomous operation of IO interfaces. However, these must be customized to be resource efficient and validated in the light of real-time constraints and the need to manage and store packets in host memory:

- *Internally,* – the structure and type of the communication system must be determined, and the performance/cost impact of buffers and priorities must be explored.
- *Externally,* – interaction patterns and appropriate accelerators must be devised. Interface latency and throughput for on-the-fly packet transfers must be considered.

### 6.7.1 Evaluation of the Message Passing System

Separating Preparation-Management (PM) and Access-Response (AR) layers (cf. Sec. 6.4) necessitates exchanging packets for TX, RX, and rate selection feedback in between these layers (and thus PEs). Time-critical channel status and packets are communicated with the SPE. With ideal point-to-point communication and unlimited buffering between tasks, throughput and timing constraints are met in a 120/200 MHz PE configuration (1x1, no TXOP continuation). As communication is mapped to NOVA's message passing, effects due to arbitration, latency, and limited buffer size must be considered.

Communication links are mapped to a **single bus** as in [207]. Each PE is both master, sending messages, and slave, receiving them. A PE therefore has one shared inbox resp. outbox abstracted by the I/O boxes of Section 5.3.2. Transfers consume 16 bus cycles for packet descriptors (PDs, 64 B) and are mutually exclusive. If the target box is full, backpressure occurs at the source, which eventually blocks the sending PE completely. With IMIX traffic (exhibiting worst-case behavior) we find this setup indeed feasible:

- *Arbitration* and *bandwidth* are not relevant due to the low load imposed by PDs.
- *Latencies* directly affect the critical response paths but can be controlled at moderate communication system frequencies (80–100 MHz).

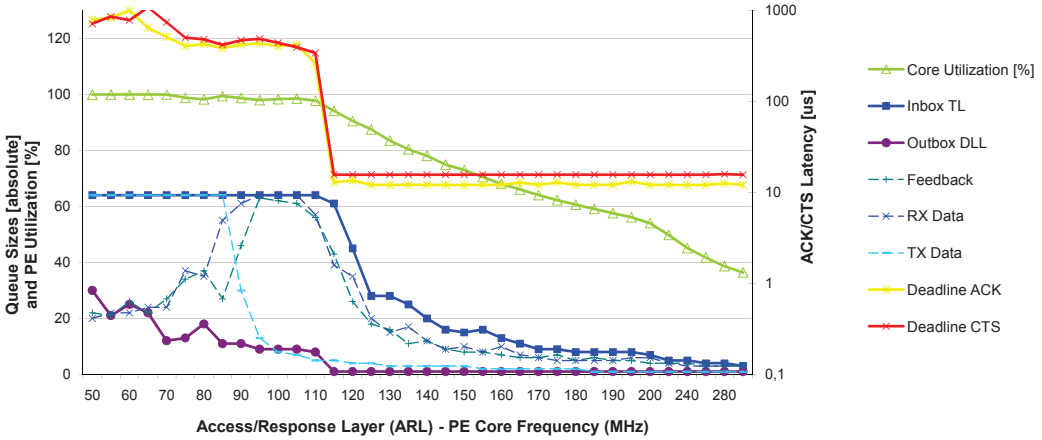



***Figure 6.15*** *– Shared I/O box sizes and core utilization (left axis) and deadlines (right axis) for a two-PE mapping: The AR-layer core frequency is varied until backlog no longer interferes with real-time operation (120 MHz) and buffer size can be reduced (180 MHz) [2].*

However, **buffer sizes** and core frequency at the PM layer and back-pressure turn out to be a critical system parameter in terms of real-time performance and cost:

- Up to 115 MHz core frequency (PML), the inbox *limit* of 64 *is reached*, which leads to back-pressure on both the host (TX Data) and AR-layer side (RX Data and Feedback).
- Consequently, the ARL core's shared outbox is not idle, which leads to *head-of-line blocking* for real-time PHY communication and thus missed deadlines.
- At 120 MHz the *load* at the PM layer *can be handled* (utilization drops). Since the PM-layer inbox does not run full any more, no back-pressure occurs at the AR layer and deadlines are met.
- Now a *trade-off* can be made between core utilization and needed buffer size. Running the PML core at 120 MHz reaches throughput requirements but requires an inbox size of 61 PDs. More reasonably, only 8 PDs are needed at 180 MHz.

In **summary**, back-pressure is very useful in between throughput-dominated PEs (such as at the PM layer) and untimed interfaces supporting back-pressure (such as Ethernet). Buffer sizes can be kept moderate (< 512 B) if some performance headroom is left or bursts and long-running tasks are avoided. Back-pressure, however, is detrimental for real-time operation at the AR layer due to head-of-line blocking or complete blocking of the core. The message passing system must thus be adjusted at least at the AR layer to (1) support dedicated real-time I/O, e.g., based on priorities as in Section 6.4.2, and to (2) discard low priorities packets rather than to block the core if outboxes are full.

### 6.7.2 Host Interface

If the WLAN MAC is coupled and shares memory with a host processor as sketched in Figure 6.16), this *host interface* must be designed carefully. The Click approach allows for a flexible split between host and MAC as long as a packet-based interface is used. Remote payload storage necessitates on-the-fly transfers during transmission and reception.

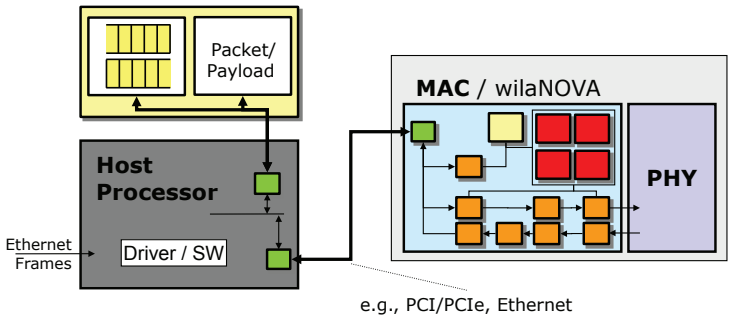



***Figure 6.16*** *– WLAN MAC system coupled with a host and external packet memory.*

**Host SW architecture.** The host caters a number of different network interfaces such as Ethernet or DSL, and application SW performs routing, user interfacing, and high-level functions. A *low-level driver* is integrated into the OS for packet hand-off and accessing the MAC via configuration APIs. The physical interface, e.g., PCI is often protected by a *Hardware Abstraction Layer (HAL)*. Polling communication at moderate rates is preferred over interrupts to reduce load on the host, as also described in [27] for a Click driver. Payload should only be moved by DMAs that must support scatter-gather operation. The memory is exclusively managed by the host in kernel buffers.

An efficient **driver interaction** requires *PD fetch/hand-off accelerators* in the MAC that aid with the following operation: The host provides packets to be **transmitted** over WLAN as a list of segment pointers in dedicated ringbuffers. Multiple queues must be supported in case QoS classification is done on the host. QoS classes may also be communicated in an extra header along with other per-packet sideband info. The MAC deploys an accelerator to poll for available packets and to prefetch headers. For transmission, the MAC's SPE fetches the payload on the fly from shared memory. Once segments are not needed any more, they are written back to a release queue on the host. Similarly, **received** packets are stored directly into host memory. The MAC buffers free segments to hide allocation latency. It can necessary to store aggregate subframes into distinct segments. After validation and reordering, the packets are handed off via a receive list on the host. The host must support scatter-gather DMA operation such that arbitrary lists of segments can form an actual packet, e.g., if headers are inserted or removed.

**Interface throughput.** For on-the-fly payload transfers, the interface to the remote packet storage must sustain the PHY's peak throughput. For example, 600 Mbit/s must be sustained for 800 $\mu$s during transmission of a maximum-sized A-MPDU. Depending on the exact PHY timing, the transfer may be started ahead of time. The state-of-the-art host/MAC setup [121] discussed in the following is connected via PCI, delivering up to 1 Gbit/s in burst mode at 33 MHz. However, the sustained throughput is dominated by **transaction latency** for small packet segments. The measured latency (1.26 $\mu$s avg., 2.82 $\mu$s worst case) is due to complex, bridged architecture in the host. The required minimum latency $\tau_i$ per burst now depends on average segment size (Fig. 6.17. Assuming segments larger than 256 B on average, the interface latency can be balanced out if a moderately sized buffer (4 kB) is added to avoid over/underflow. Systems with very high PHY rates, a very slow interface, or the ability to transmit minimum-sized aggregates at full throughput must prefetch more payload data prior to transmission. For reception, up to 8 kB are required if payload processors have very high latencies.

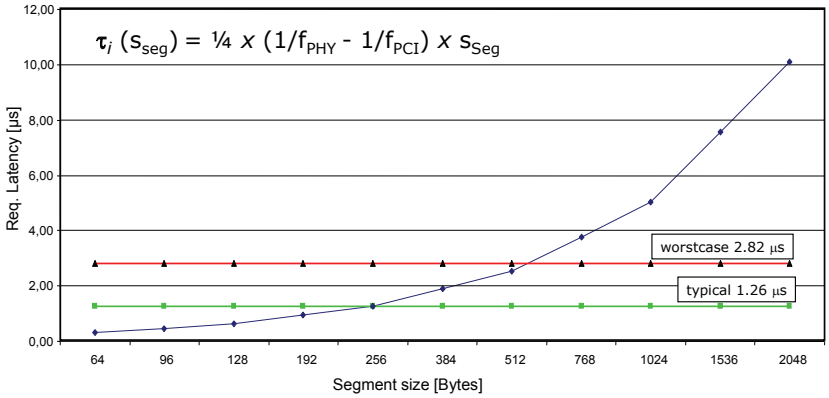



***Figure 6.17*** *– Required minimum latency $\tau_i$ (per burst) for PCI burst reads over segment size to reach 600 Mbit/s sustained throughput.*

## 6.8 Proposed *wilaNOVA* Platform Instances

Based on the preceding exploration, a candidate *wilaNOVA* platform architecture is presented. A block-diagram summarizing new building blocks and the description of packet flows underpin the validity of the chosen approach. Then, a block-level analysis of chip area costs is used to conclude on total costs of platform instances needed for the reference device configurations. The results are categorized and compared to existing work.

### 6.8.1 Platform Architecture and Operation

The proposed *Wireless LAN on NOVA (wilaNOVA)* architecture is based on the NOVA platform (cf. Sec. 5.1.1). Following universal MAC principles, it is software based and designed for flexibility to support evolving and related protocols. Specific hardware is only introduced if necessary for performance/cost reasons and targeted to generic operation rather than protocol specifics. The separation of header from payload processing is integral to our approach: Event-driven MAC protocol functions in SW operate per packet and require only the manipulation of packet descriptors and headers. By contrast, processing on packet payload data is stream-oriented and done flow through by dedicated HW. As a low on-chip memory footprint is a primary cost concern, payload is kept in remote host memory. The remaining on-chip memory is preferably allocated to shared memory to increase the flexibility of its use in the system.

The **block diagram** of wilaNOVA is shown in Figure 6.18. It features up to five processing elements (PEs) with local instruction and data scratchpad memories (IS/DS). A shared on-chip and the host memory (e.g., over PCI/e) are accessed through memory interfaces (M). Message-passing interfaces (C) transport packet descriptors (PDs) and system messages (SMGs) in between PEs and other blocks. The host interface bridges access to remote memory either memory mapped or based on SMGs. An accelerator is added for fetching and handing off PDs with the host. The specialized PE (SPE) encapsulates PHY specifics, and its modular payload processing path comprises buffers, accelerators for crypto, A-MPDU, CRC, and an optional packet parser (see Sec. 6.6).

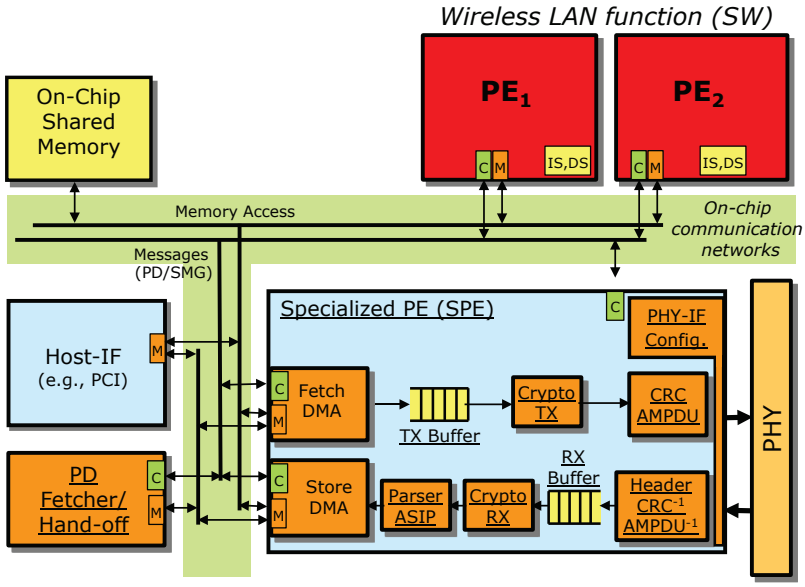



***Figure 6.18*** *– Block diagram of the resulting wilaNOVA architecture with two PEs executing the WLAN function (e.g., for a 2x2 AP). Names of conceptually new blocks are underlined.*

Once a packet is detected for **transmission**, its header is pre-fetched and a PD is generated and forwarded to SW for protocol processing. The packet then is scheduled by pushing the PD along with the transmit time and processing configuration to the SPE. A preliminary Go-Signal can be sent ahead of the actual PD to increase the SW's cycle budget. The SPE parses the PD and concatenates payload segments from the PD or from memories. Lists with extra segments can be read from shared memory or be sent in subsequent messages. The fetching process is decoupled by a FIFO and is halted by back-pressure if downstream blocks are full. Payload processing is performed on a per-packet basis as indicated by sideband information. Start of transmission and PHY configuration are delayed until the given time stamp. A-MPDU frames are sent back to back.

**Reception** is more complex due to time-critical responses and remote payload storage. Upon detection, a CCA message and time stamp is sent to SW. The SPE parses the receive context, splits A-MPDUs, and establishes the streaming protocol. Headers can be forwarded immediately to supporting PEs that aid configuring crypto and parser engines. Packet processing and storage proceeds speculatively until the CRC can be validated at packet end. Pre-buffered segments are used for storage of packets and de-aggregated subframes. System messages notify the SW about transfer completion, segment pointers, and processing results. Once SW receives the PD and packet header, it performs CRC validation, response generation, header processing, and packet reordering. The packet can be discarded from host memory, e.g, if CRC fails. Synchronization stages in SW may be needed if, e.g., check results are only made available by HW after SW processing has started. Eventually, packets are released to the host via the PD hand-off unit.

### 6.8.2 Component Costs

We start our analysis by deriving the chip area of the platform's components. The estimations are based on published work such as [207] or extrapolated from proprietary IP. We target a 65 nm high-performance process and assume that 610K NAND2-equivalent gates can be fit on 1 $mm^2$ (see, e.g., [37]). A factor of 0.65 converts from 90 nm data [189]. Memory area is a large factor and thus estimated precisely using a memory explorer tool.

An area estimate for the **PEs** is given in Table 6.18. The M4K core is synthesized for 400 MHz but should support dynamic scaling for power reasons. Memories account for up to 76 % of the area due to their high-speed configuration. Memories larger than 32 kB are composed of smaller blocks. As PD and SMG operation barely interfere in WLAN, the MP interface is optimized to a single memory and thus exclusive operation.

***Table 6.18** – Area estimations for one Processing element (PE) in 65 nm, based on [207]*

| *General-Purpose PE* | *Area [$mm^2$]* | *Remarks* |
|---|---|---|
| MIPS M4K Core | 0.135 | w/o fast multiply |
| Local Code Memory | 0.096 - 0.441 | 14 kB – 64 kB single-port (SP) RAM (highspeed) |
| Local Data Memory | 0.030 - 0.340 | 4 kB – 48 kB single-port (SP) RAM (highspeed) |
| Message Passing | 0.057 | optimized to one dual-port (DP) RAM (1280 B) |
| Other logic | 0.053 | includes memory if, DMA, and address decoder |
| **Total*** | **0.363 – 1.026** | |

The **SPE** (Tab. 6.19) also has optimized MP interfaces but needs a controller for internal configuration and SMG generation in addition. It comprises the A-MPDU/CRC blocks, logic at the MAC/PHY interface (including a timer and header extractor), and TX/RX controllers with DMAs and PD parsers. Buffers are scaled with PHY speed to decouple 50 $\mu$s of traffic from interface latency (plus 50 % in RX direction). The parser ASIP is based on a control-dominated header parser from [50]. The crypto engine is specified at 80 kGates for up to 300 Mbit/s of throughput. Considering a 32-bit data path, only 4.7 MHz to 33 MHz are required for SPE operation. Thus, all data path logic except the crypto engine can be scaled as needed and single-port high-density memories suffice.

***Table 6.19** – Estimated area for the Specialized PE (SPE), partly based on [207].*

| *Specialized PE* | *Area [$mm^2$]* | *Remarks* |
|---|---|---|
| Message Controller | 0.024 | includes 2x 128 B DP RAM |
| TX/RX CRC/A-MPDU | 0.031 | est. 4x (RX) and 1x (TX) the CRC unit* |
| MAC/PHY Interface | 0.025 | est. same as RX CRC/A-MPDU |
| Fetch/Store DMA + controller | 0.015 | includes interfaces to on-chip and host memory |
| PD Parser/Unparser | 0.038 | includes 2x 512 B DP RAM |
| TX/RX Stream Buffer | 0.028 – 0.121 | 3 kB – 20 kB SP RAM (high-density), dep. on bandwidth |
| Programmable Parser ASIP | 0 – 0.100** | includes 2kB SP RAM, not needed for Lite-MAC |
| Crypto Engine | 0.149 – 0.298 | depends on throughput & number of streams |
| **Total** | **0.310 – 0.652** | |

*) compared to [207] **) from [50], scaled from 0.45 $mm^2$ in 180 nm

The biggest **global** resource (Tab. 6.20) is shared memory, accounting for up to 0.73 $mm^2$ (140 kB). High-densitiy memory can be used due to its low speed requirements. PCI is specified at 100 kGates and delivers up to 1 Gbit/s at 33 MHz. Interconnect and arbiters for memory and MP do not contribute significantly to overall area [207].

***Table 6.20*** *– Global resources area estimation.*

| *Resource* | *Area [mm²]* | *Remarks* |
|---|---|---|
| Shared Memory | 0.089 – 0.730 | 16 – 140 kB, single-port RAM (high density) |
| Host Interface | 0.164 | PCI, up to 1 Gbit/s |
| Interconnect, Arbiters | 0.033 | for on-chip memory and MPIs |
| **Total** | **0.286 – 0.927** | |

***Table 6.21*** *– Summary of chip area costs for WLAN device configurations.*

| | | *2x2 AP* | *2x2 STA* | *2x2 AP optimized* | *1x1 AP LiteMAC* | *4x4 AP MultiMAC* | *VHT AP projected* |
|---|---|---|---|---|---|---|---|
| *PHY Speed* | Mbit/s | 300 | 300 | 300 | 150 | 600 | 1000+ |
| *TXOP Cont.* | – | yes | yes | limited | no | yes | yes |
| Number of PEs | – | 2 | 2 | 1 | 1 | 3 | 4 |
| Req. Freq. | MHz | 250/400 | <250/400 | 400 | 200 | all 400 | 3x300/400 |
| Local DS Mem. | kB | 8/20 | 4/8 | 28 | 16 | 10/14/34 | 6/4/4/48 |
| Local IS Mem. | kB | 36*/38 | 32*/38 | 64* | 44 | 36*/38/38 | 24*/14/14/38 |
| *Total (PEs)* | mm² | 1.20 | 1.06 | 0.87 | 0.67 | 1.93 | 1.87 |
| SPE / Parser | x | 1 | 0.5 | 1 | 0 | 1 | 1 |
| SPE / Crypto | x | 1 | 1 | 1 | 1 | 2*** | 2**** |
| Buffers (TX/RX) | kB | 2/3 | 2/3 | 2/3 | 1/1.5 | 4/6 | 8/12 |
| *Total (SPE)* | mm² | 0.42 | 0.37 | 0.42 | 0.31 | 0.63 | 0.65 |
| Global Memory | kB | 54 | 16 | 54 | 16/32** | 100 | 140 |
| *Total (Global)* | mm² | 0.56 | 0.36 | 0.59 | 0.39/0.47 | 0.85 | 1.00 |
| ***Total Area*** | **mm²** | **2.18** | **1.80** | **1.89** | **1.37/1.45** | **3.41** | **3.54** |
| *Total Memory* | kB | 168 | 109 | 157 | 83/99 | 288 | 318 |
| *Memory/Area* | | 58.8% | 52.4% | 60.8% | 52.5/55.0% | 62.3% | 59.4% |

*) includes 12 kB (8 kB STA) for mgmt. functions **) add. packet buffer ***) 2x crypto ****) crypto scaled

### 6.8.3 Device Configuration Costs

Based on the block-level estimates, the chip areas[12] for the reference device configurations of Section 2.3 are presented in Table 6.21.[13] The **2x2 Full-MAC access point** requires two PEs running at 400 and 250 MHz. The PEs account for more than half of the area (2.2 mm²) and memories (168 kB) for 59 % overlappingly. As a station handles only one link, 109 kB of memory and a simplified SPE suffice. Optimizing the 2x2 AP to a **single PE** is feasible but requires extra optimization effort (see Sec. 6.4.2). The area savings are moderate at 14 % due to virtually unchanged memory requirements. Local code memory is becoming very large, potentially impacting the achievable frequency of the PE's core. The **1x1 Lite-MAC** costs only 60 % of the 2x2 AP. One PE at 200 MHz suffices at the expense of reduced throughput (no dynamic TXOP). It is based on the requirements for the AR layer with additional 2/6 kB local memory for host interfacing. No parser is needed. Depending on the host interface, an on-chip packet buffer must be provisioned.

Scaling towards 4x4 Multi-MAC and VHT devices, we observe the following cost factors:

- Increasing *supported stations* from 32 to 64 impacts global memory requirements as more management structures and queues are required (see Sec. 4.2.2).

[12] Additional FPGA synthesis results are found in Section 7.3.3.

[13] For comparison with legacy systems, we also state the requirements for an **802.11abg AP** based on two PEs at 200 MHz with 4/8 kB data and two 16kB code memories that performs crypto in SW [2]. The required area is 0.95 mm² resp. 1.05 mm² with and without host interface and management functions.

- The *supported BA streams* are scaled to 32/48 according to the QoS{$x$} scenario. Thus, 18 kB resp. 35 kB are added – mainly to local data memory in the AR layer.
- Increasing *throughput*, more PDs are present in the system. The SPE's crypto engine and buffers must be scaled, and more cores are needed in a pipelined PM-layer.

The **Multi-MAC** is dimensioned as a 4x4 AP but can be reconfigured, e.g., to 3x3 plus 1x1 for operation in both 11n and legacy mode. Two AR-layer cores are now needed for response generation, as both MACs operate independently. Therefore, memory is added to the second PM-layer core, which would effectively operate as a one-PE system that shares management functions with the first PM-layer core. For a symmetric 2x2 configuration (up to 462 Mbit/s), either using CET on all cores or an extra PE would be necessary. All global resources can be shared. The SPE overhead for processing two streams independently is estimated to 20 % (generic logic area, parser is unchanged). Two crypto engines are needed for concurrent streams in excess of 300 Mbit/s.

For the **Very High Throughput (VHT)** device the PHY speed is scaled to 1 Gbit/s while leaving the MAC protocol unchanged, representing a future wireless standard (see Sec. 2.2.4). Real-time requirements are unaffected, but up to 3 PM-layer PEs are needed to sustain throughput. The PCI host interface reaches its limit. The PE's payload accelerators (e.g., crypto) are scaled accordingly, assuming twice the area. Other resources such as parsers can be duplicated as needed.

**Discussion.** The memory hierarchy remains heterogeneous throughout device configurations. Even single-PE systems distinguish local and shared memory, as the later is cheaper and can also be accessed directly by the SPE. Local memory – especially on the data side – differs greatly between cores. This could be balanced to some extent for greater generality, but fundamental differences remain: The AR-layer core performs a multitude of functions and requires many data structures locally. One PM-layer core executes management functions that require code memory but map data to shared memory. Pipelined PM-layer cores require little code and data memory.

The devices can be characterized in terms of size and memory proportions (Fig. 6.19). Memories make up at least half of the area and increase with throughput and features:

- The *Lite-MAC* is 40 % smaller than the 2x2 AP, as only essential real-time functions and half the throughput are supported. A *station* is approx. 20 % smaller due to fewer streams, less queueing, and limited management functions. Both configurations have thus relatively small memory proportions just above 50 %.
- The *VHT device* is at least 50 % bigger than the 2x2 AP due to resource scaling and additional PEs, but has the same memory proportion (ca. 59 %). This indicates a constant progression as throughput is scaled.
- The *4x4 Multi-MAC* is similar in size but dominated by memories (65 %) required for flexible and independent operation of two MACs. Scaling MAC features and function has thus a bigger impact than scaling throughput.

### 6.8.4 Comparison with Existing Systems

The device configurations are compared with existing systems (as introduced in Sec. 2.2.3) in terms of memory, performance, and chip area. Little quantitative data are available for

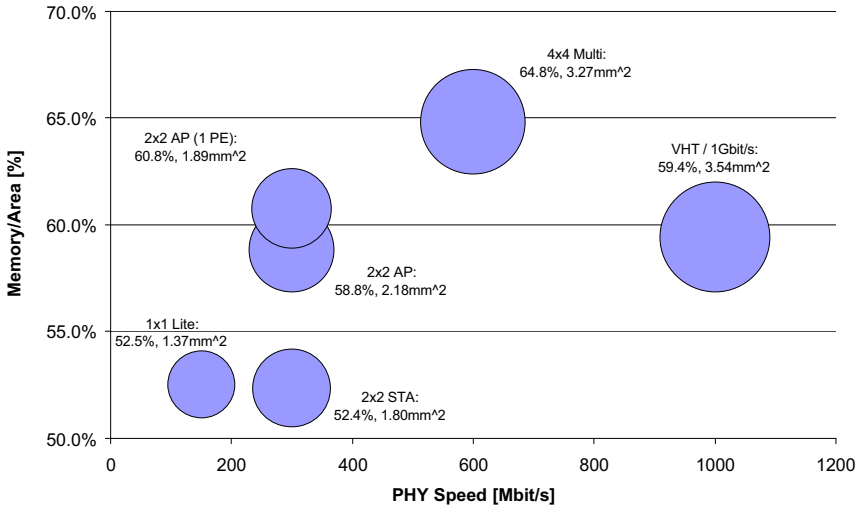



***Figure 6.19*** *– Memory/area ratios and chip area (bubble size) of device configurations.*

the MAC layer – especially of IEEE 802.11n – as many authors focus on the classical signal processing aspects of the PHY layer.

**Memory requirements** are difficult to compare due to differing function splits, host interfaces, and supported features and WLAN standard versions:

- Implementing a legacy MAC in SDL, Hannikainen [75] requires 490 kB of code and 161 kB of data memory, one order of magnitude larger than our implementation. This is due to the SDL runtime environment and the inclusion of host functions.
- An ad-hoc implementation in C [24] implements only basic features such as fragmentation and occupies about 125 kB of DSP memory. Much better is the 8-bit supplemental processor in [126], that has 8/4 kB code/data memory but only generates immediate response frames such as ACK. This is somewhat in the range of our 32-bit general-purpose implementation for 11abg.
- A fully-featured, programmable 11abg MAC by Wipro [245] uses 128 kB in station configuration. Similarly, the Intel IPW2200 firmware has 195 kB. Our legacy setup has only about 50 kB but requires packet storage and high-level functions externally.
- As the only 11n reference, a commercial Lite-MAC driver [121] has approx. 250 kB of code memory, but also covers management functions. The firmware [121] is coded in 16-bit assembly language and thus about half the size of our AR-layer code while data memory is comparable. A relatively large packet buffer is needed.

We find our application code to be better than straight-forward implementations, but fail to compete with highly optimized systems. Of course, the function kernels in our Click elements can also serve as the basis for optimized code. Total on-chip requirements in our case are moderate, as extended management functions are mapped to the host and packet buffering is reduced significantly due to on-the-fly payload handling.

The **performance** analysis showed that 400 MHz are sufficient for handling even the dynamic retransmission of frames during a TXOP in 802.11n. Legacy operation and static schedules are much simpler and can well be handled at below 165 MHz.

- Panic [179] requires 1 GHz for the 802.11a protocol with generated SDL code on a MIPS M4K. SDL requires a runtime environment and consequently fails to meet real-time constraints. Similarly, Shono [220] implements the MAC fully in SW on a PowerPC at 400 MHz but fails to meet deadlines by at least one order of magnitude.
- Also for legacy WLAN, the supplemental processor in [126] runs at 136 MHz and generates responses while channel access is done by a general-purpose core. Their analysis of the critical path and time budget concludes that only 5 $\mu$s are available for MAC processing, which is improved significantly in our approach (cf. Sec. 4.1.1).
- More recently, real-time response generation is acknowledged as a problem for flexible MACs in SDR contexts. However, only basic ACK/RTS handling is typically considered. While Nychis [172] does not give performance data, Hedde [78] successfully uses a MicroBlaze core at 100 MHz. Tan [229] achieves real time on commodity multi-core HW at 2 GHz. His main problem is not processor performance but IO latency and the need to keep a core available, similar to our findings.
- A commercial 11n Lite-MAC [121] runs at 80 MHz but relies on HW for frame aggregation and does not support dynamic TXOP continuation.

Our architecture proves superior to other fully programmable devices and academic approaches. Commercial implementations with reduced functionality or HW support are more efficient, but typically lack flexibility and a uniform, high-level programming model. None of the approaches performs dynamic frame aggregation and retransmission in SW or discusses optimizations such as Go-Signals and application-aware scheduling.

**Chip area** is the main cost factor and design driver in today's home-networking devices. It depends on the architectural approach, flexibility supported, and the maturity of the solution (see Sec. 2.2). Comparison is complicated by the high integration level that maps analog parts together with the combined digital part (MAC+PHY) on the same chip.[14]

- The fully hard-coded 11g MAC in [248] requires only 120 kGates, but excludes crypto, higher protocol functions, and buffers. Our SPE in legacy configuration is less than half in size yet does not handle protocol state and response generation.
- Excluding memories, Panic [179] states 169 kGates for his 11g setup, of which the HW accelerator is 63 kGates. Our SPE is smaller than the accelerator, but our single-PE instance would be 185 kGates without processor memories and SPE buffers. The overhead is mostly due to modularity, i.e., the message passing system.
- The highly-optimized commercial legacy MAC by Wipro [245] excluding host interfaces requires 120 kGates. Again, the overhead on our side is due to modularity and lack of optimizations, e.g., tailoring the crypto core to lower throughputs.
- Targeting a generic and simplified WLAN standard, the FPGA-based approach in [166] requires 200 kGates for the (reconfigurable) HW accelerator alone. The total of 1.34 mm$^2$ excluding memories seems rather inefficient.
- The commercial 11g MAC by Atheros [155] accounts for 43 % of the combined digital part, as indicated by the die image. The equivalent 2.5 mm$^2$ in 65 nm seem very large, but the device potentially includes complete (STA) management functions.

[14] In addition, different processes are used. Conversion factors to 65 nm for equivalent area given in mm$^2$: 0.2 from 180 nm, 0.35 from 130 nm, 0.65 from 90 nm.

- An early 11n-draft system by Atheros [185] with a 2x2 PHY and 16 BlockACK streams uses an equivalent of 12 $mm^2$ for the digital part. Analyzing the die image, 1/4 is occupied by the MAC. Considering that PCI/e is supported, the resulting area is still 20 % above our estimations for a 2x2 AP, indicating a premature solution.
- Recently, Atheros presented a more efficient 802.11n (2x2) solution [206]. The single-chip SoC uses an equivalent of 8.8 $mm^2$ for the digital part. Again, considering 1/4 as the MAC area and the PCI/e interface, our 2x2 AP solution shows a 10 % overhead.
- The commercial 11n Lite-MAC in [121] is 15 % smaller than our configuration with 16 kB of shared memory added as an extra packet buffer for better comparability.

Our own area estimations proved to be accurate and on par with both legacy and state-of-the-art devices. In legacy configuration, the SPE is smaller than typical Low-MACs. But the 11abg standard is simple and mature such that highly-optimized systems can be much smaller than our scaled-down modular platform instance. The 11n-draft system [185] shows that commercial implementations of evolving standards with short time-to-market can be immature and are outperformed by our flexible and systematic approach. Compared to more mature and recent 11n products, we estimate an overhead of 10–20 % mostly due to modularity, high-level programming model, and lack of optimization on our side. Little information is available on future devices such as the description of a 4x4 transceiver [36]. Regarding the number of BlockACK streams, Song [222] describes a gigabit system running up to 84 HDTV streams, as compared to 48 in our VHT device.

### 6.8.5 Suitability for Future Devices

Future wireless (LAN) protocols and extensions to IEEE 802.11 as discussed in Section 2.2.4 require flexible and scalable architectures. The MAC system must cope with increased throughput, MIMO, decreased response times, and extended highly dynamic functions. Decreased response times will be motivated by better protocol efficiency, i.e., reduced inter-frame gaps and faster channel access. Fast link adaption and handling of MIMO streams may be time-critical and require precise timing and concurrent processing.

The proposed architecture is well suited for such challenges, as it is based on a scalable and concurrent platform [208]. NOVA's socket interfaces and the streaming protocol of the SPE allow to modularly add or replace components. Our approach is generic and programmable. In addition, the VHT scenario has shown that scaling to gigabit throughputs is possible, and complex operations such as aggregation are already performed in realtime. We expect the following **architectural aspects** to be **impacted**:

- The number of *processing elements* can be increased for pipelining or to exploit application concurrency. Core types and frequencies can be adjusted to higher demands. Non-realtime cores with caches can be added for management functions. Similarly, *hardware accelerators* can be added or duplicated as needed. Block-level scaling, e.g., of the crypto core is beyond our scope.
- As more clients are added, the *communication* networks can be implemented by NoCs. PEs can be coupled directly to the SPEs to optimize latency. High-performance *external interfaces* such as PCI/e will be used for packet transfer. These provide multi-gigabit throughputs but have significantly larger footprints. Dedicated connections and memory ports can be used to limit the impact on the rest of the system.

- Shared *on-chip memory* requirements were moderate so far and their frequency can be further increased. *External memory interfaces* for packet buffering and code can be justified by cheap low-speed host interfaces, QoS requirements, or increased SW complexity. A multi-port memory controller limits the interference from packet transfers, while caches reduce traffic and latency for code fetches on selected cores.

## 6.9 Chapter Conclusion

This chapter presented step four of our methodology towards an efficient and flexible WLAN MAC. Following the Y-Chart, the NOVA platform was extended and explored towards optimized application-architecture mappings for WLAN – **wilaNOVA**. Keeping the reference application fixed, only mapping and platform instance were changed, while avoiding specific hardware where possible. As a result, protocol-agnostic concepts for MAC operation could be identified, and eventually the enhanced platform was scaled to our range of device configurations to assess chip area costs. Throughout the process, SystemClick enabled fast evaluation of design points and a productive exploration.

The baseline analysis showed that a fully-programmable realization of the benchmark on NOVA is infeasible and inefficient, even if expensive operations are hardware accelerated. Thus, further exploration of the design space was necessary:

- The ***MIPS M4K core*** is well suited for deeply-embedded systems, as it offers a better performance/area trade-off than more advanced cores (e.g, 24k). Running at 400 MHz in NOVA, its performance suffices for critical WLAN paths. Compiler configuration impacts code size and performance by up to 23 %, and additional trade-offs are possible at the element granularity. The ***Click*** memory overhead can be reduced by 40 %, but a small run-time overhead due to function calls remains.
- High performance requirements are caused by long-running tasks that block the resource at frame reception. Isolating the critical AR layer leads to feasible ***multi-PE mappings*** of 2–4 cores. Pipelining the PM layer for higher throughputs is better than parallelizing TX and RX processing. In addition, a systematic approach to decrease task length based on priority scheduling and exploitation of application knowledge was assessed. It can increase resource efficiency on real-time cores and be leveraged to realize single-PE solutions.
- A ***heterogeneous memory hierarchy*** is needed to trade off performance, flexibility, and cost. The consequence are access penalties from shared or remote memories that severely impair system performance, unless clock synchronization is minimized and on-chip memories are used for frequently accessed data. Element state and current packet context (PD and header) must be mapped locally to optimize throughput, but queues, enqueued PDs, and management data should remain shared to minimize private memory. Real-time performance is sensitive to A-MPDU size as PDs are enqueued in shared memory. These effects can be mitigated, if a reduced PD list is cached locally and a DMA is added to the PE. In critical cases the size and number of aggregation streams can be reduced.
- Storing packet data on-chip is expensive, and data transfers increase the impact of access contention to up to 20 %. At the same time, ***protocol and payload processing can be separated***: Protocol processing can be restricted to indirect packet operations

based on segment lists and the packet header stored in the PD. These operations must be complemented by memory management and supported by the IO interfaces (scatter-gather). Payload processing is grouped and allocated to HW accelerators at the MAC/PHY interface, where the payload is available anyway. The flow of packet data can thus be decoupled completely and storage be provisioned remotely.

- Only little dedicated hardware is needed for WLAN – encapsulated in the ***Specialized PE (SPE)***. The SPE extends NOVA's IO module by generic shared-medium MAC/-PHY interface functions (timing, PHY handling, channel state) and a ***modular payload processing path***. Blocks in this path are connected via FIFO-like interfaces and a streaming protocol. This establishes a packet abstraction in between blocks that effectively extends the modularity of Click onto such HW accelerators. Only few dedicated blocks are needed for WLAN: These payload processors (***CRC, crypto, A-MPDU***) are integrated productively into the SPE's processing path.
- Message ***communication*** can be adapted to real-time operation, if back-pressure and head-of-line blocking are controlled carefully. Trade-offs for buffer sizes, PE frequencies, and message priorities are possible such that small I/O boxes suffice (512 B). The host interface must support polling. Accelerators on the MAC side establish an autonomous, PD-based interface. PCI throughput can be limited by per-segment delays but on-the-fly payload fetch is feasible in realistic cases.

Summarizing the key benefits of our MAC architecture, the **protocol function** – including real-time critical control frame processing – can remain entirely **in SW**, and packets can be stored remotely for a **small on-chip memory footprint**. Essential for this were (as explained above) the introduction of the SPE with shared-medium MAC/PHY functions and its **modular streaming path** integrating De-AMPDU and crypto accelerators. At the system level, only a tiny DMA function, a packet-based host interface, and support for indirect packet operations is required. Precise customization of PE and the memory hierarchy is essential to guarantee throughput and real-time requirements.

The resulting wilaNOVA instances proved to be competitive with other systems, while being based on a flexible, programmable, and generic platform and following a systematic, high-level development approach. Memory was a significant area contributor (over 50 %). This ratio was almost stable when scaling bandwidth, but increased as more features were added. Assessing the **WLAN benchmark configurations**:

- The ***state-of-the-art*** 2x2 AP in Full-MAC configuration was our main design driver. It features two PEs and requires 2.2 $mm^2$ and 168 kB of memory. This is 20 % better than a comparable, early commercial 802.11n system by Atheros. If configured as a Station, the area can be reduced by 17 %.
- For an ***optimized single-PE version*** of the 2x2 AP, advanced scheduling techniques, SW and platform optimizations, and more HW support are required. The area can be reduced by 13 % and is comparable to a recent commercial 802.11n system.
- At the ***low end***, a 1x1 AP configured as Lite-MAC requires only 1.45 $mm^2$ for one PE and 99 kB of memory. It is 15 % larger than a comparable, heavily optimized commercial Lite-MAC, mostly due to modularity in our approach.
- A ***next-generation or high-end*** device in 4x4 AP configuration supporting concurrent multi-MAC operation is significantly larger at 3.41 $mm^2$, three PEs, and 276 kB of memory. Offering less features but more throughput, the VHT device is comparable in size and runs on four PEs.

*wilaNOVA* proved scalable, flexible, and inherently modular. It thus is well suited for future protocols, where challenges arise from increased throughput, decreased response times, and extended functionality. This impacts the number of PEs, communication interfaces, and the memory architecture. The architecture's potential in the context of a universal MAC and future prospects in home networking are discussed in Chapter 8.

Throughout the exploration process, **SystemClick** proved well suited for the fast and early exploration of a platform's design space based on a system model. Especially:

- Key to an indicative exploration was the precise estimation of platform overhead and accurate ***performance feedback*** (instructions, memories). The abstraction of platform services separates concerns and led to reproducible results.
- SystemClick's ***architectural models*** enabled a multi-faceted exploration and covered all performance-relevant system aspects such as the memory hierarchy.
- ***Analysis*** of real-time constraints, traces, memory accesses, protocol conformity, utilizations, and throughput exposed bottlenecks and enabled a targeted exploration.

Timing-exact simulation and extensive analysis allowed to generalize results and to debug the application. Still, the determination of required minimum frequencies proved difficult in some cases. This is a general problem of simulation-based approaches, as results depend on input and simulation time is finite.[15] Also, the choice of benchmark scenarios is critical. We focused on performance-relevant corner cases, but the scenarios are static and can only cover a fixed set of parameters, e.g., in terms of aggregation.[16]

SystemClick proved very **productive**. In addition to its architectural models and analysis capabilities, modularity and script integration accelerated generation and examination of design points. High simulation performance enabled both detailed evaluations, e.g., of tightly synchronized memory accesses and the coverage of large design axes, e.g., for searching Pareto-optimal points. A total of 3 man months (MM) was spent, including non-recurring efforts for assessment of priority scheduling, the SPE architecture, and platform services. Customization for a new protocol is estimated at **2 MM** and comprises initial HW/SW analysis, multi-processor partitioning, memory mapping and dimensioning, and communication validation. During each step, most of the time is spent on initial experimental setup to expose relevant parameters (up to 1 week) and subsequent analysis steps. Experiments/simulations can be performed within few hours.

[15] Formal approaches such as [199] can overcome this problem, but in turn are dependent on the quality of the (non-functional) application model [69] that typically does not provide a path to implementation.

[16] A discussion is well beyond the scope of this work, see e.g., [110] for an overview on *design for experiments*.

# 7 IEEE 802.11n WLAN Prototype

As a final step, a fully-functional and standard-compliant IEEE 802.11n WLAN access point is implemented. It realizes essential aspects of the proposed *wilaNOVA* architecture as identified in the preceding exploration step and leverages the WLAN reference application as device software. Since wilaNOVA follows universal MAC principles in terms of a flexible, SW-based, protocol-agnostic, scalable, and easy-to-use architecture, a generic platform will eventually become available, as is discussed in Section 8.4.

Underpinning the feasibility for WLAN and the validity of our exploration results, a video demonstration setup is presented and evaluated, e.g., in terms of resource utilizations and performance. This allows to verify our predicted results, to assess the deployment capabilities of our development approach, and to discuss extensibility.

Since ease of development and flexibility to customize the platform are important, an FPGA-based prototyping system is preferable. Especially, the MAC layer must be exposed for experimentation while providing sufficient performance. Existing solutions, however, do not support 802.11n, are not standard compliant [108], are not based on FPGAs [98], or lack performance [78]. An existing prototyping system is thus used and extended by a WLAN PHY. The *RAPTOR-X64* system [100, 188] is modular and extensible. The PCI-X card can be extended by up to six modules such as a large FPGA or other hardware as needed. A special adapter accommodates and connects an Infineon/Lantiq 802.11n WLAN development board to the RAPTOR system that is used as the PHY. The resulting WLAN prototyping setup is shown in Figure 7.1.

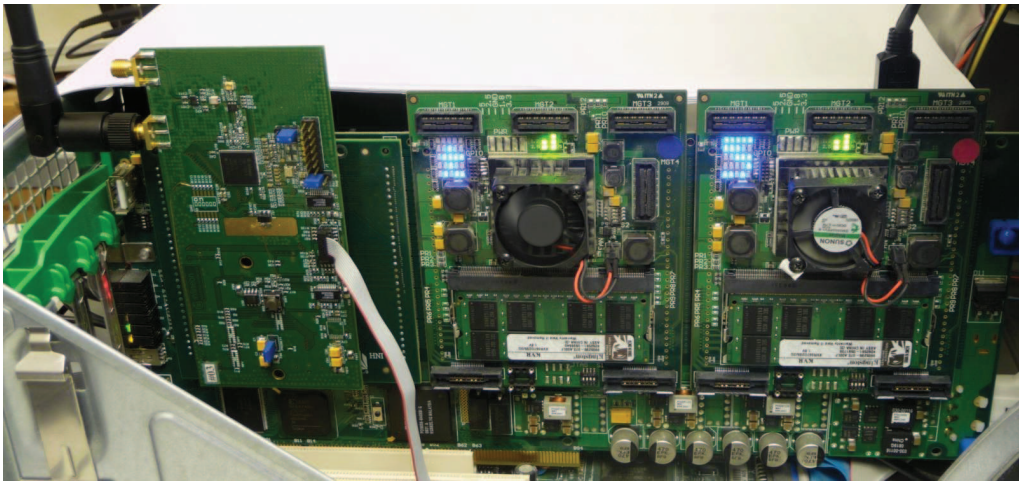


***Figure 7.1*** *– WLAN prototype with RAPTOR-X64 system and AFE/RF daughterboard.*

As the basis for realizing the wilaNOVA MAC we rely on and extend the NOVA hard- and software platform (cf. Sec. 5.1.1) and SystemClick's code generation capabilities (cf. Sec. 5.2.4). The reference application can thus be reused as device firmware. The system is limited to 1x1 configuration due to the given PHY but is standard compliant and able to operate standalone. Steps to deployment in the following include MAC implementation according to identified principles, adjustments due to the given PHY, board bring-up, demo and application setup, and performance optimizations.

## 7.1 wilaNOVA Prototype System Realization

The RAPTOR-based WLAN prototype comprises two Xilinx Virtex-4 FX100 FPGA modules, the Infineon board connected via a proprietary adapter module, and an Ethernet module as shown in Figure 7.2. All modules connect over direct left/right interconnection paths, of which the RAPTOR provides 80 per module and direction. The board is hosted in a standard PC and configured through a serial programming cable. Modules can be accessed through a PCI/on-board local bus in addition.

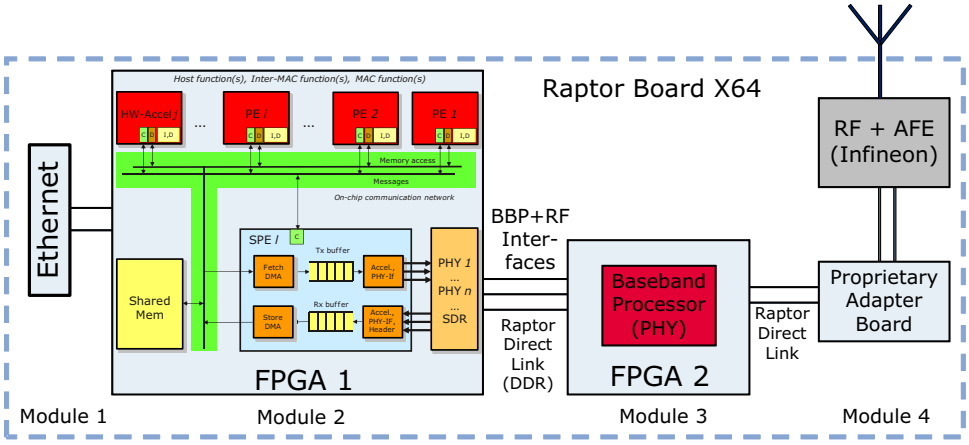



***Figure 7.2*** *– Block-level overview of the WLAN prototyping board.*

The **wilaNOVA MAC** is implemented on the module 2 FPGA, leveraging two hard-wired PowerPC cores and on-chip BlockRAM as *Processing Elements (PEs)*. These PEs perform protocol processing by running an extended NOVA operating system (OS) and an adjusted version of the reference application. The CRACC framework is retargeted to *powerpc-eabi* GNU tools, and SW including boot-up code is directly initialized into BlockRAMs. The PowerPCs are clocked at 320 MHz, but their subsystems including local memories (64 kB code and 32 kB data) are restricted to 160 MHz due to FPGA limitations. Shared memory, buses, and the SPE run at 80 MHz. Since no external memory is available, packet buffering is provisioned in shared on-chip memory (300 kB). This memory can be accessed externally for debugging purposes. The *Specialized PE (SPE)* interfaces with and configures the physical (PHY) layer. An Ethernet interface serves as a distribution system through an existing, emulated MAC-IO block.

The **PHY layer** consists of three parts: The *Baseband Processor (BBP)* performs digital signal processing for en- and decoding packets. It is connected to the *Analog Frontend (AFE)*,

which converts to and from analog signals. The *Radio Frontend (RF)* amplifies these signals and mixes them to and from the target channel frequency in the 2.4 GHz band. Up to two antennas can be connected. The PHY layer is distributed over two modules, since the Infineon WLAN chip does not allow direct access to the MAC/PHY interface. Instead, the chip's BBP part is emulated on an FPGA (module 3) while the chip itself is operated in a test mode that provides a direct digital interface to the AFE.

### 7.1.1 The wilaNOVA MAC Implementation

An integral part of the wilaNOVA MAC is the *Specialized PE (SPE)* and the encapsulated **MAC/PHY interface**. Message passing, DMAs, and CRC calculation could be reused from NOVA's Ethernet IO module, but data handling was improved to support packet headers in packet descriptors (PDs). Especially for shared-medium protocols such as WLAN the following extensions are vital:

- *Channel feedback and robustness* – Feedback to the PEs is provided via time-stamped system messages (SMGs), indicating channel state changes (CCA in WLAN), collisions, and successful transmissions. Robustness is increased by a receive (RX) trigger unit that can cope with incomplete packets and erroneous receive contexts.
- *Timing* – A 32-bit time stamp is available system-wide, in PDs, and to schedule transmissions at the interface. A two-stage mechanism initiates transmissions after a SMG is received, while packet data can be provided later (Go-Signal).
- *PHY control* – Trigger signals indicate started, completed, and ongoing transfers. In addition to direct BBP control (TX context), register-defined sequences are issued prior to and after a transmission for AFE/RF configuration. In-band RX contexts including RX rate are parsed into PDs.

The SPE's payload path is organized following the **modular flow-through streaming protocol** of Section 6.6.1. Compactors are needed after DMA and DCRC blocks to adjust byte enables. The A-MPDU accelerator could be integrated easily, because virtually no changes were necessary to other modules. In fact, only the streaming header is changed to render a burst of smaller packets transparently to a continuous aggregate and vice versa. In receive direction, the packet parser must ensure that the last frame of (even corrupted) aggregates is always marked properly for SW processing.

The **OS software** extends NOVA for improved performance, communication, and packet operations. This affects PD and system message (SMG) formats. For example, PDs feature a 32-bit time stamp and flags compacted to 12 B for signaling RX and TX contexts of the PHY. SMGs are extended to handle token communication, CCAs, and Go-Signals. The scheduler supports *ScheduleAt* operation and checks IO and expired timer in turns to reduce latency. PDs are pre-allocated. Packet operations include:

- *Packet headers* to be kept locally in the PD to reduce access latencies. The function `packet_header(n)` ensures at least $n$ byte to be available, returning a local pointer. Additional content is fetched as needed. Packet modifications (push/pull) only have local impact if possible. If the payload is needed in one segment, the `packet_data()` function swaps out the header resulting in a normal packet. Application designers should always prefer header access. IO elements must ensure that a) the header is available on input and b) that the output format conforms with the interface.

- Fast generation of *packet copies* for retransmissions under timing constraints. The original `packet_clone()` function copies and re-allocates both packet payload and descriptor, leading to long response times. It is often sufficient, however, to reference or copy only the PD, for which flags are introduced. Since copies share the payload with the original packet, the `SHARED_PAYLOAD` flag must be set and changes to PD and thus header are still possible. Packet payload may only be modified or killed by the original packet owner (`MY_PAYLOAD`). Alternatively, a reference counter is maintained such that the PD may only be killed after no more references exist. This is also used for token communication.

### 7.1.2 PHY Integration and Bring-up

A number of adjustments were made necessary by the specific choice of PHY and prototyping setup. Together with AFE/RF board bring-up, this proved very time consuming.

The **wilaNOVA MAC** is responsible for PHY configuration and control. The PHY is configured by SW at startup via dedicated interfaces encapsulated in the SPE: a bus for the BBP and a serial *Four-Wire Bus (FWB)* for the AFE/RF. The BBP features 200 configuration registers, e.g., for channel timing and *Automated Gain Control (AGC)*. The AFE/RF has 80 registers, e.g., for gain control and channel frequency. Receive and transmit modes are signaled directly to the BBP. TX packet contexts are programmed through registers, while the RX context is received in-band with the packet data. Data transfer is always triggered by the BBP. Registers need to be switched in the AFE/RF by the SPE before and after every transmission, impacting timing due to the slow serial interface.

The **BBP** part of the WLAN chip was available as RTL that had to be optimized for the FPGA. For example, 802.11b coding was removed to reduce area. Fine-tuning features are selected statically to increase operating frequency. The MAC/PHY timing differs from Section 4.1 in that the TX context (packet length, rate) is required immediately at packet start, which affects response times (see Sec. 7.3.2). Altogether, 93 % of the FPGA's logic resources are occupied and timing-closure for 80 MHz is achieved. The BBP is still limited to 20 MHz channels and normal guard intervals, resulting in 65 Mbit/s PHY rate.

The **AFE/RF** development board is connected through a proprietary adapter directly to the BBP FPGA (approx. 50 signals). The chip is operated in a dedicated FPGA test mode via an external JTAG connector. It is controlled through the FWB, and digital I/Q data is exchanged in parallel (20 bit). After optimizing the board, packet loss could be almost eliminated ($< 1\,\%$) and a sensitivity of -71.0 dBm is achieved at 54 Mbit/s OFDM modulation. This is sufficient for our demonstration setup.

**Integration** of the four modules includes a common reset and clocking concept and consideration of inter-module interfaces. The system reset is triggered by the MAC, causing a hard reset in the RF chip. The system clock is generated by the RF chip, forwarded to the MAC, scaled to 80, 160, and 320 MHz, and re-distributed throughout the system. The Ethernet module is clocked by its external PHY. For physical MAC/BBP interfacing, output data is registered at the FPGA's pads, then sampled with a skewed 80 MHz clock on the receiving end, and finally synchronized through register stages. The digital I/Q interface is strobed and operates at 160 MHz. The strobe is directly derived from the AFE's output clock and delayed for two cycles to account for slow signal slew.

## 7.2 Demonstrator Setup

The system is set up as shown in Figure 7.3 for demonstration and measurement purposes. The WLAN system serves as a standalone WLAN access point that is connected via Ethernet to a distribution system on a PC. Up to 16 wireless stations can be dynamically connected to demonstrate QoS and A-MPDU aggregation for video and background traffic applications. Multiple videos are streamed at bitrates from 1 Mbit/s to 20 Mbit/s (HDTV) using VLC media player over the UDP protocol. The tool *jperf* is used for throughput evaluation and TCP/UDP background traffic generation.

The WLAN application graph was re-modeled for prototypical deployment (starting from an 802.11g configuration) and then extended for standalone 802.11n operation and customized to the intended setup. Standard changes as compared to an early draft standard regarding BlockACK operation could be reflected easily in SW. All adjustments could be made within few weeks and included the following aspects:

- *Management* – Standalone operation necessitates standard-compliant beacon generation, association management, and BlockACK negotiations. As a central lookup facility the StationInfoBase (SIB) is mapped to shared memory.
- *Application* – Includes packet classification for STA/TID (e.g., based on destination), QoS paths per TID (=AC), and address filtering. Headers are translated from/to Ethernet. Both legacy and 11n rates are supported and dynamically adjusted for retransmissions. Direct routing to other wireless stations is provisioned.

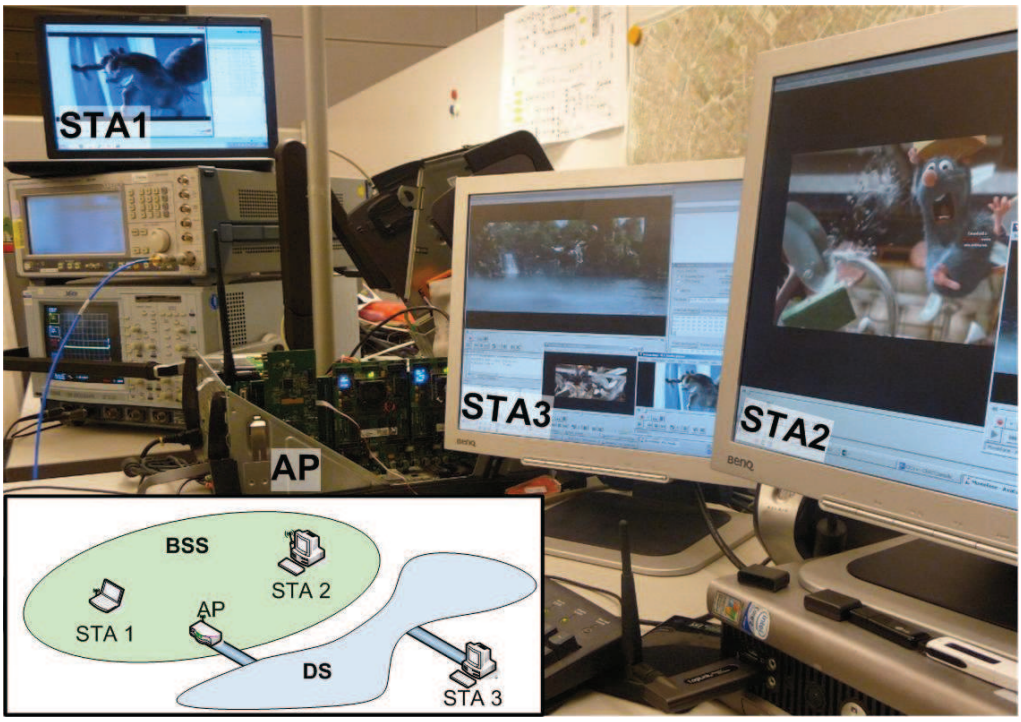



***Figure 7.3*** *– Wireless LAN demonstrator setup with AP prototype and commercial STA cards.*

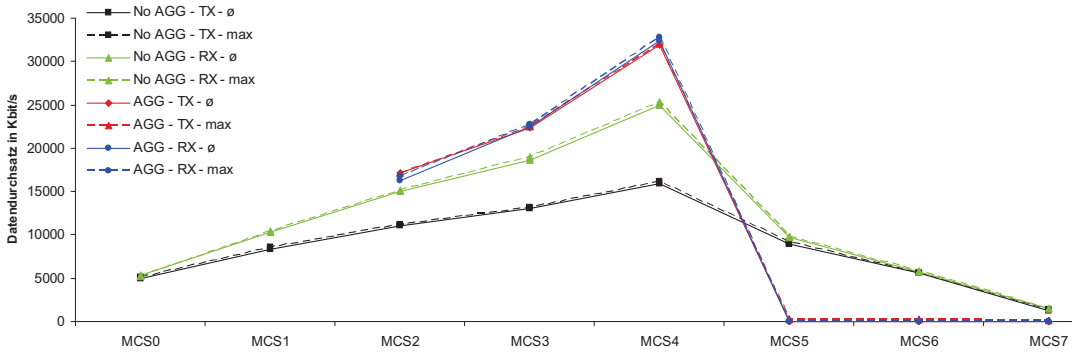



***Figure 7.4*** *– Throughput for different mcs settings with and without A-MPDU aggregation.*

- *Aggregation* – A fixed number of A-MPDU streams is available, which are assigned dynamically to negotiated sessions. Aggregates are scheduled statically and both implicit and explicit BAR are supported. BlockACK policy is immediate.

Altogether, the setup proved stable and reliable, underpinning the feasibility of the SW-based approach. Only few limitations exist that are due to our rapid prototyping approach.[1] The setup has been demonstrated successfully at Infineon/Lantiq, University of Dortmund, and in the context of the EU-funded OMEGA project.

## 7.3 Results and Performance

### 7.3.1 Network Performance and Resources

The demonstrator setup works reliably with commercial and standard-compliant 802.11n WLAN cards, which proves that all protocol constraints are met by the prototype. Standard compliance was additionally checked with protocol analyzers. The maximum *mcs* rate that performs reliably is 4 (i.e., 39 Mbit/s) due to lack of fine-tuning in the PHY. This is acceptable, as the MAC layer is in the focus of this evaluation.

Measured **network performance** further verifies our setup. The throughput is given in Figure 7.4 and is within expected bounds considering channel congestion in an office environment. A peak performance of 16 Mbit/s is achieved for non-aggregated traffic in TX direction. In RX, up to 25 Mbit/s are reached, because the commercial WLAN card drops RTS/CTS protection dynamically to increase throughput. A-MPDU aggregation proves very effective (up to 32 Mbit/s). The latency is in the low microsecond range with very little jitter. Evaluating the AP in a two-station **video** setup, up to four streams (2-4 Mbit/s) altogether can be transmitted in parallel from and to the AP without A-MPDU. Adding aggregation, bidirectional transmission of two HDTV streams (10 and 15 Mbit/s) is possible. QoS features are verified by applying additional background traffic.

[1] TXOP continuation and dynamic BlockACK length adjustments are too demanding for FPGA and have been discarded. Since the PHY rate is limited to 65 MHz, aggregates cannot comprise more than 12 subframes. A-MSDU aggregation is both less efficient and less demanding than A-MPDU and thus has been omitted. Packet payload allocation is SW-based in order to maintain flexibility and reduce latency.

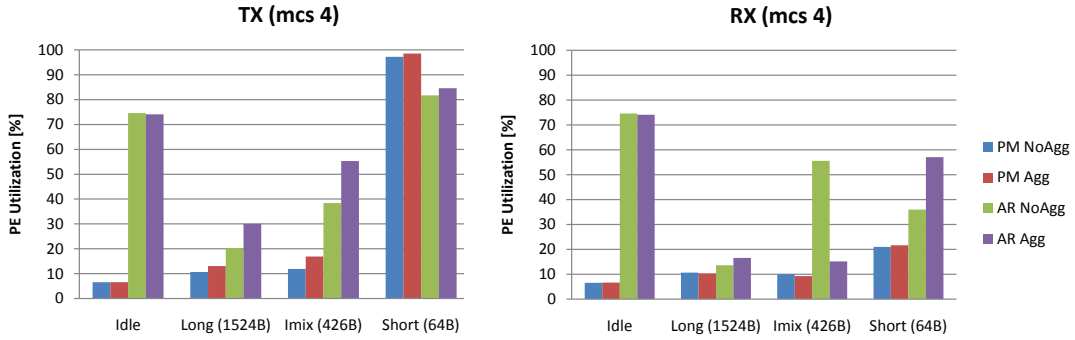



***Figure 7.5*** *– PE utilization (PM and AR layers) with and without A-MPDU aggregation (Agg.) for different packet sizes in TX and RX direction.*

**PE utilization** is a critical internal performance metric (Fig. 7.5). The high *Idle* utilization is caused by an immature channel access implementation that constantly checks for state updates. Utilization is uncritical once packets saturate the channel. With decreasing packet length, the AR layer first is more demanding, as aggregation leads to more throughput and more complex processing. If the system transmits minimum-sized frames, the throughput at the PM layer limits performance. In RX direction, utilization is uncritical even for minimum-sized frames. For *IMIX* traffic the PM layer has a large headroom, as expected for only 30 Mbit/s. The AR layer is rather sensitive to the number of packets in aggregates and thus can handle higher throughputs as well. Other **system parameters** that proved **critical** were shared memory utilization, queue sizes, and message passing. However, this can be attributed to the prototypical FPGA implementation (e.g., non-optimized memory management in SW, only 300 kB packet buffer).

### 7.3.2 Response Times

Response times are critical for standard-compliant behavior. This means that SIFS (16 $\mu$s) must be guaranteed on the channel for all responses. The **PHY latency** on the channel relative to the MAC is 8 $\mu$s until a packet is fully received by SW and of 3 $\mu$s after a transmission is triggered by SW. The latter comprises forwarding to the SPE ( 0.5 $\mu$s), PHY configuration (1.5 $\mu$s), and RF power-up (1 $\mu$s). This leaves 5 $\mu$s for SW to process incoming packets and to trigger the response transmission using a *Go-Signal* system message (see Sec. 4.1.1). Unfortunately, the PHY is not optimized for MAC latency and requires the complete TX context (rate, length) with the Go-Signal. Then, the packet's data, i.e., the PD is only required 16 $\mu$s after the Go-Signal – as expected.

All **response times** are met as shown in Table 7.1. Since A-MPDU scheduling is static, (immediate) BlockACK PD generation and channel access remain as the most critical tasks. A number of changes were made necessary by above-mentioned PHY limitations (TX context) and non-optimized code (e.g., Click overhead, OS functions):

- *Application re-structuring* – Response generators (e.g., ACK) are put close to the input. A-MPDU members are forwarded directly to ReorderBuffers to avoid backlog and improve BlockACK generation. Channel access is decoupled from scheduling.

***Table 7.1*** *– All requirements for Go-Signal delay and PD delay after Go-Signal (< 16 µs) are met (measured non-intrusively with a logic analyzer).*

| *Constraint* | *Requirement (Go-Sig.)* | *SW delay + MAC/PHY delay* | *Total (Go-Sig.)* | *PD delay* |
|---|---|---|---|---|
| DATA->ACK | 16 µs | 3 µs + 11 µs | 14 µs | 2 µs |
| CTS->DATA* | 16 µs | 2 µs + 11 µs | 13 µs | 4 µs |
| BlockACK* | 16 µs | 2 µs + 11 µs | 13 µs | **16 µs** |
| Channel Access | 5 µs | 2 µs + 3 µs | **5 µs** | 2 µs |

*) with optimized *Go-Signal Generator* element (SW)

- *Decreasing modularity* – The *send_go_signal()* function is called directly to avoid Click overhead. Some functions such as address checking and CRC flag validation are merged into neighboring elements. Some classifications including session-ID are done by specialized elements rather than standard classifiers.
- A *Go-Signal Generator (SW)* element performs a quick classification of received packets and schedules, e.g., BlockACK frames (where size is fixed). If the CTS-to-DATA constraint is critical, the EDCA can pre-program packet length and rate such that the Go-Signal can be generated immediately as well.

### 7.3.3 FPGA Performance and Resources

The wilaNOVA MAC is realized on a Virtex4 FPGA with PowerPC 440 cores. Since NOVA originally runs synchronously at 25 MHz and targets ASICs, many components had to be re-timed and **optimized for FPGA**. Given the MAC/PHY interface clock of 80 MHz, the cores run at 320 MHz and their subsystem including memories is optimized to 160 MHz. The latter necessitated restriction of data memory size (32 kB) as well as fine-tuning of address nets and BlockRAM organization. Additional register stages are needed in the PE subsystem (MPI, shared memory). This is reflected in total **memory access latency** (see Tab. 7.2). Special consideration of synchronization cycles in SystemClick's memory models (cf. Sec. 5.5.3) thus proved justified, as they account for up to 44 % of total latency.

***Table 7.2*** *– Measured shared memory access latency for PowerPC-based PE.*

| *Delay reason* | *Delay (actual clock) [cyc.]* | *Actual clock [MHz]* | *Delay (CPU clock) [cyc.]* |
|---|---|---|---|
| Store / load | 1 | 320 | 1 |
| Sync to 160 | 0–1 | 320 | 0–1 |
| Sync to 80 | 0–1 | 160 | 0–3 |
| Arbitration | 1 | 80 | 4 |
| **Sum (store)** | **n.a** | **n.a.** | **5–9** |
| BRAM latency | 1 | 80 | 4 |
| Register to CPU | 1 | 160 | 2 |
| **Sum (load)** | **n.a.** | **n.a.** | **11–16** |

**FPGA resources** are shown in Table 7.3. Comparison with ASIC gates and thus with our estimates of Chapter 6 is very difficult. Instead, we normalize to a MicroBlaze softcore (from [233]). While both the total system (5X) and the PE subsystems (0.5X) are reasonably sized, the SPE excluding memories and encryption is 2–3X bigger than a MicroBlaze. This is due to its non-optimized implementation (e.g., PD generation uses a 512-bit register) but can be optimized as needed. **Code size** proved critical for the AR layer (close to 64 kB,

***Table 7.3*** *– FPGA resources utilized by a wilaNOVA MAC comprising two PowerPCs with 64/32 kB local memory, three 4 kB buffers in the SPE, and 300 kB shared on-chip memory.*

| *Component* | *Resource* | *absolute* | *of FPGA total* |
|---|---|---|---|
| Total MAC | Slice/Flipflops | 7623 | 9 % |
| | Slice/LUTs | 12162 | 14 % |
| | BlockRAMs | 276 | 73 % |
| | IOBs | 217 | 37 % |
| | BUFGs | 17 | 53 % |
| | DCMs | 4 | 33 % |
| | PowerPCs | 2 | 100 % |
| | MicroBlaze equiv. (FF/LUTs)* | 4.9/5.3 | n.a. |
| PE subsystem (hard-wired PowerPC) | Slice/Flipflops | 841 | 1 % |
| | Slice/LUTs | 700 | 1 % |
| | BlockRAMs | 52 | 13 % |
| | PowerPCs | 1 | 50 % |
| | MicroBlaze equiv. (FF/LUTs)* | 0.5/0.3 | n.a. |
| SPE (MAC/PHY If) | Slice/Flipflops | 3348 | 3 % |
| | Slice/LUTs | 6352 | 7 % |
| | BlockRAMs | 10 | 2 % |
| | MicroBlaze equiv. (FF/LUTs)* | 2.1/2.7 | n.a. |

*) MicroBlaze Virtex4 resources, 75 MHz, no caches, no FPU: 1554 FFs/2296 LUTs (from [233])

almost 50 % more than expected) and less tight on the PM side (approx. 45 kB). However, the increase is due to the need for a higher SW optimization level (-O3) to compensate for reduced PE performance in the FPGA (below 200 MHz effectively). In addition, *init* code is not swapped out and Click overhead is not removed. Debugging code and *assertions* account for almost 10 kB. Altogether, the estimated values are still within reach.

## 7.4 Discussion

### 7.4.1 Expected Results

**Observable quantities** were in the range expected from SystemClick simulations. Net throughput approaches the theoretical limit and latencies depend on traffic class and Block-ACK operation (as in Sec. 3.5.1). On-chip buffers and queues proved sufficient, their capacities can be derived from simulation. Advanced packet operations (headers) yielded the expected performance gain if used carefully by the developer. The overall load on the system was uncritical as expected, and PE utilizations were low in general even after packet classification was added. However, because of the high number of CCAs in the real system, channel access (EDCA) increased load in the idle state.

Direct comparison to **predicted costs and performance** is difficult. Memory and area differ because the prototype is a standalone system on an slow(er) FPGA with non-optimized HW blocks. Response times required SW optimizations due to limited PE performance and non-optimized OS and Click overhead. While SystemClick had identified critical paths and their requirements correctly, the optimization itself proved tedious due to an increase in complexity after partly breaking with modularity. The benefits of a modular SW architecture thus became apparent: despite the performance penalty, clearly defined and separated functions are easier to adjust, debug, and verify.

Many unexpected issues appeared when the system interacted with the **real world**, i.e., the given PHY, wireless channel, and other (commercial) WLAN stations. A critical case were HDTV streams with an invalid target port. This caused short *ICMP* packets in the reverse direction, simultaneously flooding the system with very long and very short packets. This packet burst and the lack of dedicated memory management led to a feedback loop between the two PEs that overloaded the message passing system.[2] The principle problem had been predicted correctly by the analysis in Section 6.7.1. However, this indicates that (1) packet bursts must be considered better in benchmark definitions, (2) the memory allocation facet must be considered during exploration, and (3) dedicated memory management is preferable. Other issues include:

- The PHY signaled *erroneous packets* and spurious CCAs. Especially, the *Signal Field)* containing packet length is unprotected (802.11g), invalidating our streaming protocol. This was fixed by padding/pruning data directly at the MAC/PHY interface.
- Unexpected or initially *misinterpreted protocol behavior* (e.g., BlockACK operation) and even *standard changes* (e.g., BlockACK bitmap layout) occurred. Similarly, previously neglected management aspects required refinement. The *endianness* of the PowerPCs caused a mismatch with simulation for the A-MPDU bitmap. However, these changes could be easily fixed in software (see next section).

### 7.4.2 Approach and Productivity

The available **tool set** ranges from SystemClick to single-core ISS over RTL simulation to FPGA emulation. All tools cover the full system function at different abstraction levels and trade off timing accuracy, platform representation, and ease of use. This enables a seamless development process. **SystemClick** provides accurate performance feedback (cf. Sec. 5.6.2) and thus yields a-priori complexity bounds. These upper bounds give confidence that functions are feasible in principle. During development, SystemClick proved very useful for quick verification of application changes. Its full use for debugging was hindered by minor inconsistencies, e.g., the profiling tool chain had not been adjusted for PowerPC. Also, code refinement for wilaNOVA's platform specifics (e.g., split transaction memory allocation) and optimizations (e.g., Go-Signal generator in SW) were not disciplined enough, leading to diverging code bases. Finally, differing intrinsics of simulation and target CPU such as endianness must be overcome by enforcing portable code. However, no principle obstacle exists to refining the SystemClick representation as close to the target as needed.

The total deployment time for the prototype was close to two years and can be estimated to **19 effective man-months (MM)**, as outlined in Table 7.4. A closer look reveals the usefulness of our approach and tools:

- The time to *extend, refine, and adjust* the application SW to the standalone setup and the real PHY was only **4 MM**. The platform's programming model (Click) proved efficient and enabled fast changes (e.g., only one week for the actual demo setup). Initial efforts were small, since the reference model can be used directly as device firmware if implemented in a performance-indicative and thus reasonably efficient way. A protocol standard change in the A-MPDU bitmap could be fixed within hours.

[2] As a resolution, the message system's capacity was increased, memory management was optimized to buffer segments in shared memory and to frequently perform compaction, and scheduling was tuned to alternate between SMGs and PDs.

***Table 7.4** – Demonstrator development timeline.*

| *Time / Date* | *Phase / Milestone* | *wNOVA** | *Board** | *WLAN** |
|---|---|---|---|---|
| 2007 / 2008 | System modeling + exploration /w SystemClick | (5) | | (6) |
| End of 2008 | Planning for adapter board and PHY | | 0.5 | |
| March 2009 | HW/SW platform development, SPE, NOVA OS | 4 | 0.5 | |
| – Oct. 2009 | application extension and restructuring | | | 2 |
| Oct. 2009: | AFE/RF chip available, RTL simulation works | | | |
| Oct. 2009 | Board bring-up and debug (BBP+RF, MAC/PHY IF, | 2 | 6 | |
| – Sept. 2010 | FPGA-to-FPGA, clocking, Ethernet), | | | |
| | application refinement, A-MPDU accelerator | | | 2 |
| April 2010: | First packet capture (TX) | | | |
| Sept. 2010: | WLAN (11g) connection stable (PER < 1 %) | | | |
| Oct. 2010 | A-MPDU integrated, performance optimization | | | 1 |
| – Nov. 2010 | Demo setup and evaluation | | | 1 |
| Dec. 2010: | Prototype fully operational (11n, A-MPDU, QoS) | | | |

*) Estimated effective man-months (MM), team of up to 3 students

- An additional **2 MM** were spent on *performance optimization* and *dedicated HW development* in the SPE, most of which can be attributed to the A-MPDU accelerators. The integration benefited greatly from the modular streaming protocol in the SPE, leaving other (generic) blocks untouched.
- *Platform development*, *bring-up* (Board), and *debug* accounted for **13 MM** altogether (Sec. 7.1). Consequently, debugging capabilities must be of primary concern (e.g., a platform-independent `printf` or a SW-triggerable on-chip logic analyzer).

### 7.4.3 Prototyping the Next Protocol

Revisiting the **universal MAC (UMAC)** concept, we evaluate how productively the prototype can be used for related protocols. The time for platform development (6 MM) can be considered a one-time effort, including a large initial overhead for targetting the FPGA, porting to PowerPC, setting up interfaces, and establishing a debug infrastructure. With the SPE, extended OS functions (e.g., packet header), and generic support for shared-medium protocols with unreliable PHYs, a capable and generic platform becomes available for reuse. The SPE's modular streaming protocol facilitates future integration of HW blocks. Platform changes are only expected in the case of significantly differing real-time, performance, or channel access requirements.

**Recurring costs** include application deployment and bring-up of PHY and board. The deployment time for a related protocol application depends on its complexity and the quality of the reference model. If no performance optimization (HW/SW) as for A-MPDU is needed, recurring costs can be reduced to 4 MM. Board bring-up can be significant, but mostly depends on expertise and support (which was minimal in our case for the PHY, hence the large effort). At the same time, the RAPTOR FPGA prototyping system proved very useful, and 2–3 MM should suffice for integrating and configuring a well-supported PHY.

Altogether, faster application deployment, easier platform customization, and avoiding FPGA/PHY pitfalls is possible. Reusing the prototyping board when **targeting another protocol similar to WLAN**, a deployment effort **in the range of 6–9 MM** seems feasible if the following points are considered:

- Getting *platform and board/PHY* working is aided by better tool support (e.g., on-chip debug, RTL), but current tools lack support for SW analysis. Debugging should be planned ahead, i.e., leaving headroom in terms of performance and code memory. A well supported and proven-to-work PHY is definitely beneficial.
- *Refinement and optimization* must be done in a disciplined way, relying on versioning and regression testing. Platform specifics must be hidden (e.g., using macros). Optimization by breaking modularity is difficult and should thus be avoided.
- *SystemClick* should be used throughout the complete development process, refining the platform and application in SystemC towards the target system. This includes message passing, memory allocation, and the SPE model.

## 7.5 Chapter Conclusion

The presented FPGA-based WLAN prototype is amongst the first to demonstrate the IEEE 802.11n protocol on a fully-programmable architecture. Based on the RAPTOR prototyping system, it integrates the **wilaNOVA MAC**, an emulated baseband processor, an AFE/RF chip, and Ethernet. The MAC function is directly generated from the reference application, and all protocol functions including real-time control frame processing run in SW on an extended NOVA OS. The MAC implements a generic MAC/PHY interface and the SPE's modular streaming protocol in HW. A module for packet splitting/concatenation as needed for A-MPDU (de-)aggregation is integrated. The system is adjusted to the given PHY and had to be optimized due to performance constraints of the FPGA.

The **demo setup** comprises a fully-functional, stand-alone, and standard-compliant WLAN access point that interoperates with off-the-shelf wireless cards. Streaming of multiple HDTV videos and QoS could be demonstrated successfully. The achieved throughput of 15 Mbit/s at 39 Mbit/s PHY rate can be increased to over 30 Mbit/s with A-MPDU aggregation. All observable quantities are in good compliance with the expected results from SystemClick simulation. Response times in SW are met as expected, but necessitated optimizations in terms of application re-structuring, breaking with modularity, and acceleration of the Go-Signal. The (non-optimized) total MAC system occupies 9 % of the logic resources of a Xilinx Virtex 4 FPGA, comparable to 4–5 microblaze soft cores.

Both the chosen approach and the available tools proved **productive**. Re-using the reference, the application was deployed in 6 effective man months (MM). SystemClick and its programming model enabled fast adjustments of the system function for the demo setup and to counter standard changes. SystemClick provides seamless support during deployment and performance optimization, if SW is refined in a disciplined way and the platform representation is kept consistent (which is highly recommended). Board/PHY bring-up was tedious and complicated by lack of support (7 MM). Initial platform development effort was significant (6 MM), but future deployments will benefit from the implemented platform, generic MAC/PHY interface, and debugging infrastructure.

Our concept towards a flexible MAC proved feasible, and a powerful **platform for shared-medium protocols** becomes available. The prototype fully exposes the MAC layer of 802.11n for experimentation or related protocols. Given a portable system reference for a new shared-medium protocol, MAC deployment can be expected to be within 6–9 MM, depending on application complexity and the level of support for the PHY.

# 8 Thesis Conclusion

*Media Access Control (MAC)* plays a central role in communication devices for shared-medium protocols commonly found in home networks such as *Wireless LAN (WLAN)*. With ongoing protocol evolution, the MAC layer has grown and still grows in complexity and requirements (e.g., real-time). It now poses a unique challenge to a system's design, since adaptability as required for evolving protocol families cannot be provided by today's conventional, dedicated devices.

Instead, programmable platforms are needed that provide flexibility in terms of the system's function and that can be quickly adapted to related protocols. Disciplined approaches to the development of such platforms are vital in terms of productivity and to trade off architectural features in the light of tough market competition. These new MAC architectures must be protocol agnostic, flexible, easy to use, scalable, and yet efficient. Development for the MAC layer must address the challenges posed by application complexity, real-time requirements, and the need for productive development while allowing seamless exploration of architectural trade-offs.

The contribution of our work is threefold, as summarized in the following: A comprehensive system-level development framework and methodology, which was shown to be both applicable and productive in the context of home networking protocols (Sec. 8.1). The development of a real-world MAC system from an executable IEEE 802.11n WLAN benchmark as a demanding representative of such protocols (Sec. 8.2). The exploration and optimization of an architectural platform that allows to develop and prototype competitive MAC systems, which are fully-programmable from a high-level description (Sec. 8.3). Altogether, a universal MAC (UMAC) architecture and development approach emerges with great potential, not only in home-networking products but also with the advent of cognitive and software-defined radios (Sec. 8.4).

## 8.1 Methodology and Tools

Our work addresses the lack of established methods for the development of programmable MAC platforms that leads to unproductive development cycles and inefficient designs. By successfully developing a WLAN system from initial protocol analysis to prototypical deployment, the selected **application-driven approach** was proven to be feasible and well suited in terms of applicability to such shared-medium MAC protocols. Starting with a performance-indicative application model poses a large initial effort and requires early involvement of SW developers – unlike traditional system engineering approaches. However, the methodology helped tackling the problem of application complexity in a systematic and productive way and yielded efficient results:

- The ***reference application*** captured domain knowledge and helped recognizing essential protocol features and performance-relevant aspects. It was vital for early analysis

of real-time and memory size requirements, the two dominant factors for cost and feasibility of WLAN system. Also, it served as an executable specification aiding exploration and design.

- Relying on a ***modular, single-source model*** avoided reimplementation and was key for early exploration on abstracted hardware resources. The Click language with our extensions regarding timing and communication proved ideal for modeling complex, time-critical interactions as needed for shared-medium MAC protocols. Supporting Click as a programming model opens a direct path to implementation and eases maintainability, which is often neglected in other approaches.
- The ***Y-Chart*** abstraction proved useful during exploration, since both application and architecture needed frequent optimization and changes, e.g., for assessing feature/cost trade-offs (aggregation). Fast turn-around and explicit mapping (scheduling, memories) quickly led to feasible and competitive solutions.

The exploration and development framework **SystemClick** follows the Y-Chart in combining Click with early SystemC performance models. We contributed to SystemClick [9] and extended it (1) to fully complement the application-driven methodology, (2) to counter the challenges associated with shared-medium MAC protocols, and (3) to cover the associated design space better, leading to improved systems:

- ***Modeling and Mapping*** – Extending the expressiveness of Click and its mapping abstractions proved essential for exploring efficient application-to-architecture mappings. Our graphical tool (*CliMMT*) increases productivity and has been well received by concept engineers at Infineon [8].
- ***Automated Performance Feedback Loop*** – A feedback loop integrating *profiling, simulation, and analysis* was introduced [4] that puts the application into the focus – accounting for its complexity. Precise feedback (computation, communication, memories) was vital for verification of real-time constraints. User-defined granularities and orthogonalizing behavior from (platform) performance allowed indicative platform exploration and optimization with fast turn-around.
- ***Architectural Models*** – SystemClick's expressiveness was extended for *memories* [5] and *shared communication* [2]. Access penalties from such shared resources due to latency, arbitration, and synchronization that impact (real-time) performance can be reflected. Two different abstractions allow coupling with access-accurate SystemC models or grouping accesses to abstracted resources. The later is 4X more precise than previous approaches and up to 150X faster than access-accurate models, allowing quick searching of large design spaces.
- ***Simulation Performance and Accuracy*** – Above-mentioned abstractions (performance, memories) allowed speedups of three orders of magnitude over ISS after optimization. Compared to RTL simulation, up to five orders of speedup are achieved even if memories are considered (unlike many other approaches). The accuracy is sufficient to indicatively reflect design changes, e.g., to the memory hierarchy [7].

The extended framework is seamless and comprehensive and thus increases productivity. It is based on appropriate abstractions for application, architecture, and mapping above the C language and RTL, which enables early and productive exploration, better debugging, and fast simulation. Despite the framework's lightweight nature it shows good quality of results. It thus overcomes limitations of other frameworks due to lack of abstraction, lack of flexibility, and extensive tool requirements. Based on SystemC, it is extensible and

especially the newly added memory facet allows direct coupling with, e.g., TLM models of communication infrastructure.

The importance of **productivity** in an industrial project setting motivated a closer analysis of this aspect, as is summarized in Table 8.1. First-time efforts total 36 effective man months (MM). Exploration benefited from our new methods, but feasibility was analyzed before automated feedback was available. Prototypical deployment was tedious due to lack of support (PHY layer). Recurring efforts are expected to be significantly lower (as will be further discussed in Sec. 8.4). Changes to the model setup, protocol extensions (as, e.g., in [1]), or developing a closely related protocol are a matter of days or few weeks. The overall productivity gain from our system-level approach cannot be underestimated: System engineering efforts (in industry) are estimated at 3–6 man years each for specification and implementation. In fact, increasing device complexity might render development infeasible altogether without system-level methods.

***Table 8.1*** *– Estimation of total real initial efforts and expected recurring efforts for developing a different shared-medium protocol.*

| *Development phase* | *Initial efforts (WLAN) [MM]** | *Exp. rec. efforts [MM]* |
|---|---|---|
| Domain analysis | n.a. | n.a. |
| Application model and benchmark definition | 8 | 4–6 |
| Architecture-independent analysis / feasibility | 6 | 1 |
| Design exploration and definition | 3 | 2 |
| Prototypical deployment | 19 | 6–9 |
| Total | 36 | 13–18 |

*) Estimated effective man-months (MM), team of up to 3 students

## 8.2 Wireless LAN Benchmark

To the best of our knowledge, this is the first published evaluation of the **IEEE 802.11n MAC protocol** in a system engineering context with product-relevant scope. Our methodology and framework provided the systematic foundation and associated tools for the following development steps:

- A ***domain analysis*** revealed relevant features (e.g., aggregation), device configurations (e.g., MIMO), and principle architectural concerns (function split, packet memory). Flexibility in function and architecture proved a key requirement for evolving and future protocols.

- A comprehensive ***reference application for the 802.11n MAC*** layer was developed [6]. Unlike other models, it covers all performance-relevant aspects (e.g., aggregation) and is implemented in a performance-indicative way. Such a reference was not available before, and it has already been well received for realistic evaluation of protocol extensions [1]. Benchmark scenarios were devised that expose corner cases and add detail to configuration, environments, and measurement.

- An ***architecture-independent analysis*** – extended on the benchmarks deriving memory and real-time requirements. The need for a programmable, concurrent, and real-time capable platform and support for its exploration was identified.

A programmable platform template was optimized for MAC operation and customized for our range of reference WLAN device configurations. No other approach so far has considered the full 802.11n MAC function on a fully-flexible architecture supporting a high-level programming model and has explored trade-offs with quantitative results.

- ***Range of devices*** – The *state-of-the-art* 2x2 access point (AP) as a Full-MAC was our main design driver. It comprises two processing elements (PEs) and 168 kB of memory, totaling a chip area of 2.2 $mm^2$ (in 65 nm). A *station* requires 17 % less area. At the *low end*, a 1x1 AP as a Lite-MAC requires only 1.45 $mm^2$ for one PE and 99 kB of memory. A *next-generation or high-end* device in 4x4 AP configuration supporting concurrent multi-MAC operation is significantly larger at 3.41 $mm^2$ (three PEs, 276 kB).
- The results are ***competitive*** with commercial systems, despite being based on a flexible and programmable platform and a high-level development approach. The 2x2 AP is 20 % better than a comparable early commercial 11n system by Atheros. More recent system are approx. 10 % better, which could be matched if our AP was optimized to one PE. An overhead of up to 15 % is incurred over highly optimized low-end 1x1 devices mostly due to modularity.
- Memory was a significant ***cost factor***, accounting for 50-60 % of the chip area, followed by the processing elements. Scaling function (e.g, Multi-MAC) is more expensive than scaling throughput. Stations should optimize code size (i.e., Lite-MAC) while access points must minimize on-chip data and packet memory. PEs can be run below 400 MHz, which is realistic for the selected architecture, and below 200 MHz, if dynamic aggregation is relaxed.

## 8.3 MAC Architecture Exploration and Prototypical Deployment

The basis for the competitive results of the preceding section was the careful consideration of architectural trade-offs in order to reduce chip costs while maintaining flexibility. During this **design space exploration**, instances of a modular platform (NOVA) were optimized for MAC processing and customized for WLAN leveraging SystemClick [3, 5]. Starting from a fully-programmable baseline instance and a precise SW profile of the reference application, the exploration followed universal MAC principles in avoiding dedicated HW and favoring protocol-agnostic solutions where possible:

- Required *HW accelerators* (A-MPDU aggregation, CRC, crypto, DMAs), *multi-processor mappings* (2–4 PEs), and *SW optimizations* (e.g., Click overhead) were identified.
- The *memory hierarchy* was optimized by allocating small portions of memory locally (25 % performance gain), separating packet storage, and identifying trade-offs for large number of aggregation streams (prefetch).
- System *communication* has to be prioritized for real-time responses, and a packet-based *host interface* supporting scatter-gather operation is required.

No further protocol-specific accelerators were shown to be necessary. However, advanced indirect packet operations and memory management must be provisioned efficiently. This leaves the **protocol function** – including real-time critical control frame processing and response generation – **almost entirely in software**.

Leveraging existing platform features (message passing, heterogeneous memories), the following architectural extensions proved key to the (WLAN) MAC system, including the introduction of a **Specialized PE (SPE)** at the MAC/PHY interface:

- Strict ***separation of packet header and payload*** is enforced, countering shared access penalties, contention caused by payload moving, and limited on-chip buffers. This includes header-specific extensions to the platform OS and better store/fetch mechanisms in the SPE. Payload processing is encapsulated in the SPE where payload is available naturally.
- Click's modularity and packet passing semantics are extended towards HW blocks in the SPE's processing path by establishing a ***modular streaming protocol***. The SPE thus is easy to customize and extend (e.g., A-MPDU) and can be leveraged by high-level synthesis and reconfiguration.
- ***Generic MAC functions*** for shared medium and real-time operation are added, including precise time-stamping (TX/RX), channel status updates, PHY configuration mechanisms, and increased MAC/PHY interface robustness.

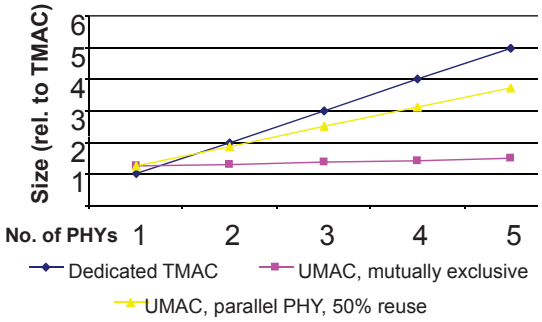
The resulting MAC-optimized platform is scalable, flexible, and inherently modular. It is thus well suited for future protocols, where challenges arise from increased throughput, decreased response times, and extended functionality. In addition, it provides a foundation for adaption to other MAC protocols.

An FPGA-based **IEEE 802.11n prototype** was developed, verifying for the first time the feasibility of a software-based MAC that comprises complex features such as A-MPDU aggregation and real-time handling of control frames. Our framework proved very helpful in customizing, optimizing, and verifying the application in the prototype. The resulting system is fully programmable and open. This allowed to correct a standard change within hours, but also makes it ideal for MAC-layer experimentation: For example, a fine-granular rate selection algorithm could be evaluated within one week, which is not possible with closed or less flexible systems. The stand-alone demo setup with HDTV videos was successfully demonstrated at Infineon and for the OMEGA project.

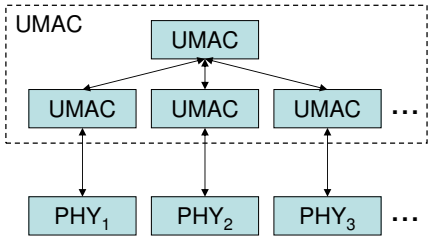
## 8.4 Towards a Universal MAC

The importance and heterogeneity of MAC functions in residential home gateways and the increasing trend towards flexible radio systems has motivated us to strife for a more universal and homogeneous MAC approach. Such an approach must be protocol agnostic, flexible, scalable, efficient, and easy to use (cf. Sec. 1.4). Following these principles throughout this thesis, we have developed a modular HW/SW platform for the WLAN MAC protocol that is fully programmable and supported by a seamless framework. Since WLAN is at least representative for other shared-medium home-networking protocols in terms of real-time and throughput requirements and complexity, the combination of these aspects lays the foundation for a truly **universal MAC (UMAC)** [3]:

1. ***Protocol function in software*** – Almost the complete protocol function relates to header processing and can be performed in software, as has been shown for WLAN including real-time critical control frames. The platform is programmable in Click with its extended application library for shared-medium protocols, optimized expressiveness, and graphical front-end in a productive way.




**(a)** Area advantage of a UMAC over dedicated TMACs for exclusive and parallel setups.




**(b)** UMAC-based architecture.

***Figure 8.1*** *– The promise of a Universal MAC (UMAC) architecture.*

2. ***Modular hard- and software platform*** – The underlying architecture template is inherently modular and scalable. In addition, the SPE's streaming protocol extends Click's packet abstraction and modularity to a flow-through payload processing path. The resulting platform can thus be customized to the requirements of other protocols and dedicated accelerators can be integrated as needed. Fast deployment is further aided by generic MAC functions for shared medium protocols (e.g., channel status, timing, efficient packet operations).

3. ***Development and exploration framework*** – Our methodology and tools are comprehensive and productive and have been proven to yield efficient, programmable MAC solutions. Productive exploration of the design space (multiprocessor mappings, memory, communication) enables fast customization. High-level synthesis of Click elements into SPE hardware blocks would further increase productivity.

The **development efforts** for implementing a new shared-medium protocol are estimated in Table 8.1. Taking advantage of the established flow and platform, total time to prototypical deployment drops to one third (13–18 MM). This depends on application complexity and the maturity of the used PHY. Protocols that are more closely related or protocol extensions (e.g, future versions of IEEE 802.11) can be developed significantly faster.

The potential of a UMAC system **supporting multiple physical layers (PHYs)** and associated MAC protocols is sketched in Figure 8.1a. A maximum overhead of 25 % for a single UMAC over a dedicated *TMAC* is assumed. Now, estimating that mutually exclusive MAC operation would add only around 10 % to a scaled UMAC's chip area for interfacing logic per PHY, a UMAC setup with two PHYs is already 30 % smaller than two separate TMACs. In a setup with concurrent (parallel) MAC operation, the UMAC is smaller starting with three PHYs. Here, it is assumed that each additionally supported PHY adds only 50 % to the chip of a single UMAC instance, since, e.g., host interfacing, external memory interfaces, pins, selected accelerators and management functions can be shared.

There are at least two promising **areas of deployment** where we expect benefits from an open, homogeneous, and flexible UMAC architecture as sketched in Figure 8.1b:

- *Customer Premises Equipment* – More efficient designs arise from unified, system-level trade-off explorations, from component sharing, and from dynamic load distribution on memories, processors, and queues. Subsystem integration is improved, as, e.g., bottlenecks and overhead can be reduced. Better maintainability and re-use are achieved,

decreasing time to market and non-recurring costs for engineering. In fact, new and previously unattractive markets may become available altogether. At the same time, the chip area penalty from modularity and programmability are becoming negligible. Since analog components and pads do not shrink as fast as digital logic, the digital MAC part will account only for a very small fraction of overall area in future feature sizes (< 45 nm).

- *Fully software-defined radio* – Eventually, a single fully SW-based UMAC can control a number of PHYs connected, e.g., through a bus-like structure in a time-multiplexed fashion. Maximum flexibility is reached in combination with software-defined radio [192] or reconfigurable PHYs, potentially aided by reconfiguration of HW blocks in the MAC's modular SPE module.

## 8.5 Directions of Future Research

Our work has been successfully applied to WLAN. Even more potential arises from the advent of flexible communication systems and increasing momentum in *Electronic System Level (ESL)* methodologies:

- *Embedded Software* – Click has shown its large potential as a modeling language, with the distinction between graph and element level holding great promise. It could, however, be more formalized, e.g., in terms of element prerequisites and modifications to packets. The element description (or embedded software in general) should make requirements more explicit (avoiding profiling) and provide better mapping abstractions (e.g., for data structures).
- *SystemClick* – The architectural modeling capabilities e.g., for communication could be extended or complemented by other models. All existing facets could be leveraged for power estimation, e.g. based on utilization ratios. Further uses of SystemClick include performance modeling of existing architectures to evaluate the feasibility of a given application-to-architecture mapping, or SW-centric virtual prototypes. In both cases, the memory models can be extended as common shared resource predictors and must be complemented by cache models.
- *Platform architecture* – Accelerator modules in the SPE can be generated using high-level synthesis, which extends code generation from Click elements onto hardware. Such hardware accelerators could furthermore be reconfigured dynamically as required for software-defined radio applications.
- *Protocol development* – This work has focused on WLAN. Future work may address the intersection of platform support and protocol development. Additional functions can be realized easily, either for fast deployment of other protocols or for evaluation of protocol extensions to existing protocols.
- *UMAC* – Fully programmable solutions are feasible, but require more research, e.g., in terms of power saving. The UMAC should be further evaluated in terms of fully flexible software-defined and cognitive radios.

# Acronyms

**AC** WLAN: Access Category

**ACK** WLAN: Acknowledgement

**AES** Advanced Encryption Standard

**AFE** Analog Front End

**AGC** WLAN: Automated Gain Control

**AIFSN** WLAN: Arbitration Inter-Frame Space Number

**A-MPDU** WLAN: Aggregated Mac Protocol Data Unit

**A-MSDU** WLAN: Aggregated Mac Service Data Unit

**AP** WLAN: Access Point

**API** Application Programming Interface

**ARL** Access-and-Response (AR) Layer

**ASIC** Application-specific Integrated Circuit

**ASIP** Application-specific Instruction Set Processor

**BAR** BlockACK Request

**BBP** Baseband Processor

**BlockACK** WLAN: Block Acknowledgment

**BSS** WLAN: Basic Service Set

**CCA** WLAN: Clear Channel Assessment

**CET** Click Element Threading

**CliMMT** Click Modeling and Mapping Tool

**CPE** Customer Premises Equipment

**CPI** Cycles-per-Instruction

**CPTE** Cycles Per Timing Event

**CRACC** Click Rapidly Adapted to C Code

**CRC** Cyclic Redundancy Check

**CSMA/CA** Carrier Sense Multiple Access with Collision Avoidance

**CTS** WLAN: Confirm-to-send

**DCF** WLAN: Distributed Coordination Function

**DIFS** WLAN: Distributed Interframe Space

**DLS** WLAN: Direct Link Support

**DMA** Direct Memory Access

**DSL** Digital Subscriber Line

**EDCA** WLAN: Enhanced Distributed Channel Access

**ELF** Executable and Linking Format

**ESL** Electronic System Level

**FCFS** First-Come First-Serve

**FIFO** First-In-First-Out

**FPGA** Field Programmable Gate Array

**FSM** Finite State Machine

**HCCA** WLAN: Hybrid Coordination Function Controlled Channel Access

**HD(TV)** High Definition (Television)

**HT** WLAN: High Throughput

**HW** Hardware

**ICMP** Internet Control Message Protocol

**IF** Interface

**IMIX** Internet (Traffic) Mix

**IO** Input/Output

**IP** Internet Protocol

**IPTV** Internet Protocol Television

**IR** Intermediate Representation

**IS/DS** Instruction Side- / Data Side-

**ISS** Instruction Set Simulator

**JTAG** Joint Test Action Group

**KPN** (Kahn) Process Network

**LoC** Lines of Code

**MA** Memory Adapter

**MAC** Medium Access Controller / Media Access Control

**MCS** WLAN: Modulation Coding Scheme

**MIMO** Multiple Input Multiple Output

**MM** Memory Manager

**MM** Man Month

**MoC** Model of Computation

**MPI** Message Passing Interface

**MP-SoC** Multiprocessor System-on-Chip

**NAV** WLAN: Network Allocation Vector

**NoC** Network-on-Chip

**NOVA** Network-Optimized Versatile Architecture

**NRE** Non-recurring Engineering

**OFDM** Orthogonal Frequency Division Multiplexing

**OS** Operating System

**PCF** WLAN: Point Coordination Function

**PCI(/e)** Peripheral Component Interconnect (Express)

**PD** Packet Descriptor

**PDU** WLAN: Protocol Data Unit

**PE** Processing Engine

**PHY** Physical (Layer)

**PLC** Powerline Communications

**PML** Preparation-and-Management (PM) Layer

**QoS** Quality-of-Service

**RDG** WLAN: Reverse Direction Grant

**RF** Radio Frequency

**RG** Residential Gateway

**RIFS** WLAN: Reduced Interframe Space

**RM** SystemClick: Resource Manager

**RTL** Register Transfer Level

**RTOS** Real-time Operating System

**RX** Receive / Reception

**SAP** WLAN: Service Access Point

**SDL** Specification and Description Language

**SDR** Software-Defined Radio

**SDRAM** Static Dynamic Random Access Memory

**SIB** Station Info Base

**SIFS** WLAN: Short Interframe Space

**SMG** System Message

**SoC** System-on-chip

**SPE** Specialized Processing Engine

**SW** Software

**TCP** Transmission Control Protocol

**TDMA** Time Division Multiple Access

**TID** WLAN: Traffic Identifer

**TLM** Transaction-level Modeling

**TP** Throughput

**TX** Transmit / Transmission

**TXOP** WLAN: Transmission Opportunity

**UMAC** Universal MAC

**UML** Unified Modeling Language

**UWB** Ultra Wide Band

**VHDL** Very Large Scale Integration Hardware Description Language

**VHT** WLAN: Very High Throughput

**VoIP** Voice-over-IP

**WAN** Wide Area Network

**WEP** WLAN: Wired Equivalent Privacy

**wilaNOVA** Wireless LAN on NOVA

**WLAN** Wireless LAN

**WMM** WLAN: Wifi Multimedia

**WPA** WLAN: Wi-Fi Protected Access

# List of Figures

# List of Tables

# Author's Publications


[1] O. Hoffmann, F.-M. Schaefer, R. Kays, C. Sauer, and H.-P. Loeb. Prioritized medium access in ad-hoc networks with a SystemClick model of the IEEE 802.11n MAC. In *IEEE 21st International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC)*, 2010.

[2] H.-P. Loeb, M. Gries, and C. Sauer. Implementing a software-based 802.11 MAC on a customized platform. In *Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference (CCNC)*, 2009.

[3] H.-P. Loeb, C. Liss, U. Rückert, and C. Sauer. UMAC - a universal MAC architecture for heterogeneous home networks. In *The International Workshop on Wireless and Optical Networks (WI-OPT 2009), at International Conference on Ultra Modern Telecommunications (ICUMT-2009)*, 2009.

[4] H.-P. Loeb and C. Sauer. SystemClick - efficiently exploring the performance of programmable packet processing platforms. In *18th European SystemC User's Group Meeting*, 2008.

[5] H.-P. Loeb and C. Sauer. Exploration of embedded memories in SoCs using SystemC-based functional performance models. In *Forum on Specification Design Languages (FDL)*, 2009.

[6] H.-P. Loeb and C. Sauer. A modular reference application for IEEE 802.11n wireless LAN MACs. In *Proceedings of the 2009 IEEE International Conference on Communications (ICC)*, 2009.

[7] H.-P. Loeb and C. Sauer. Fast SystemC performance models for the exploration of embedded memories. In D. Borrione, editor, *Advances in Design Methods from Modeling Languages for Embedded Systems and SoCs*, volume 63 of *Lecture Notes in Electrical Engineering*, pages 73–92. Springer Netherlands, 2010.

[8] H.-P. Loeb, C. Sauer, M. Gries, and S. Dirk. Graphical component-based application modeling and code generation for MP-SoCs. *Infineon Embedded Software Days (IESD)*, 2006.

[9] C. Sauer, M. Gries, and H. P. Loeb. SystemClick – a domain-specific framework for early exploration using functional performance models. In *Proceedings of the 45th ACM/IEEE Design Automation Conference (DAC)*, 2008.

[10] C. Sauer, H.-P. Loeb, and C. Teerapat. Performance of an 802.11n software MAC. Technical report, Infineon Technologies, Access Communication Solutions, 2007.

# Bibliography


[11] S. Abdi, G. Schirner, I. Viskic, H. Cho, Y. Hwang, L. Yu, and D. Gajski. Hardware-dependent software synthesis for many-core embedded systems. In *Proceedings of the 2009 Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2009.

[12] G. Agha. *Actors: a model of concurrent computation in distributed systems*. MIT Press, Cambridge, MA, USA, 1986.

[13] Agilent Technologies. JTC 003: mixed packet size throughput. *The Journal of Internet Test Methodologies*, pages 16–18, Sept. 2004.

[14] D. Akhmetov. 802.11n: Performance results of reverse direction data flow. In *Proceedings of the IEEE 17th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2006.

[15] T. Arpinen, P. Kukkala, E. Salminen, M. Hännikäinen, and T. D. Hämäläinen. Configurable multiprocessor platform with rtos for distributed execution of uml 2.0 designed applications. In *Proceedings of the conference on Design, automation and test in Europe (DATE)*, 2006.

[16] K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, S. Williams, and K. Yelik. The landscape of parallel computing research: A view from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California at Berkeley, 2006.

[17] R. Balani, C.-C. Han, R. K. Rengaswamy, I. Tsigkogiannis, and M. Srivastava. Multi-level software reconfiguration for sensor networks. In *Proceedings of the 6th ACM & IEEE International conference on Embedded software (EMSOFT)*, 2006.

[18] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara. *Hardware-software co-design of embedded systems: the POLIS approach*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[19] F. Balarin, Y. Watanabe, H. Hsieh, et al. Metropolis: an integrated electronic system design environment. *IEEE Computer*, 36(4):45–52, 2003.

[20] J. R. Bammi, W. Kruijtzer, L. Lavagno, E. Harcourt, and M. T. Lazarescu. Software performance estimation strategies in a system-level design tool. In *Proceedings of the eighth international workshop on Hardware/software codesign (CODES)*, 2000.

[21] A. Banerjee and A. Gerstlauer. Transaction level modeling of best-effort channels for networked embedded devices. pages 77–88, 2009.

[22] F. Baskett, K. Chandy, R. Muntz, and F. Palacios. Open, Closed, and Mixed Networks of Queues with Different Classes of Customers. *Journal of the ACM*, 22(2):248–260, 1975.

[23] L. Benini, R. Hodgson, and P. Siegel. System-level power estimation and optimization. In *Proceedings of the 1998 international symposium on Low power electronics and design (ISLPED)*, 1998.

[24] R. Bernasconi, S. Giordano, A. Puiatti, R. Bruno, and E. Gregori. Design and implementation of an enhanced 802.11 MAC architecture for single-hop wireless networks. *EURASIP Journal on Wireless Communication Networks*, 2007(1):33–33, 2007.

[25] D. P. Bhandarkar. Analysis of memory interference in multiprocessors. *IEEE Transactions on Computers*, (9):897–908, Sept. 1975.

[26] G. Bianchi, A. Di Stefano, C. Giaconia, L. Scalia, G. Terrazzino, and I. Tinnirello. Experimental assessment of the backoff behavior of commercial ieee 802.11b network cards. In *26th IEEE International Conference on Computer Communications (INFOCOM)*, 2007.

[27] A. Bianco, R. Birke, D. Bolognesi, J. Finochietto, G. Galante, M. Mellia, M. Prashant, and F. Neri. Click vs. Linux: two efficient open-source IP network stacks for software routers. In *Workshop on High Performance Switching and Routing (HPSR)*, 2005.

[28] O. Blaurock. A SystemC-based modular design and verification framework for C-model reuse in a HW/SW-co-design flow. In *Proceedings of the 24th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 2004.

[29] A. Bobrek, J. Paul, and D. Thomas. Shared resource access attributes for high-level contention models. In *Proceedings of the 44th ACM/IEEE Design Automation Conference (DAC)*, 2007.

[30] A. Bobrek, J. Pieper, J. Nelson, J. Paul, and D. Thomas. Modeling shared resource contention using a hybrid simulation/analytical approach. In *Proceedings of Design, Automation and Test in Europe (DATE)*, 2004.

[31] N. Bombieri, F. Fummi, and D. Quaglia. System/network design-space exploration based on tlm for networked embedded systems. volume 9, pages 1–32, New York, NY, USA, 2010. ACM.

[32] R. Bosman, J. Lukkien, and R. Verhoeven. An integral approach to programming sensor networks. In *Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference (CCNC)*, 2009.

[33] S. Boukhechem and E.-B. Bourennane. Tlm platform based on systemc for starsoc design space exploration. In *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2008.

[34] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. Advances in network simulation. *Computer*, 33(5):59–67, 2000.

[35] Broadcom. BCM4323x family - Intensi-fi® XLR media family. Product Sheet, accessed online, 2010.

[36] A. Burg, S. Haene, M. Borgmann, et al. A 4-stream 802.11n baseband transceiver in 0.13 CMOS. *Symposium on VLSI Circuits*, pages 282 –283, 2009.

[37] D. Bursky. Behind the numbers: Are FPGAs usurping ASIC design investigations and design starts? *Chip Design Magazine*, 2007.

[38] C.-B. Chae, A. Forenza, J. Heath, R.W., M. McKay, and I. Collings. Adaptive mimo transmission techniques for broadband wireless communication systems. *Communications Magazine, IEEE*, 48(5):112 –118, 2010.

[39] B. Chen and R. Morris. Flexible control of parallelism in a multiprocessor pc router. In *In Proceedings of the 2001 USENIX Annual Technical Conference (USENIX)*, 2001.

[40] C.-H. Chen and F.-F. Lin. An easy-to-use approach for practical bus-based system design. *IEEE Transactions on Computers*, 1999.

[41] M. K. Chen et al. Shangri-la: Achieving high performance from compiled network applications while enabling ease of programming. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2005.

[42] E. Cheung, H. Hsieh, and F. Balarin. Framework for fast and accurate performance simulation of multiprocessor systems. In *IEEE International High Level Design Validation and Test Workshop (HLVDT)*, 2007.

[43] E. Cheung, H. Hsieh, and F. Balarin. Memory subsystem simulation in software TLM/T models. In *Proceedings of the 2009 Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2009.

[44] CoWare. CoWare PlatformArchitect. `www.coware.com/products/platformarchitect`, accessed Jan. 2009.

[45] M. Damm, J. Moreno, J. Haase, and C. Grimm. Using transaction level modeling techniques for wireless sensor network simulation. In *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*, 2010.

[46] D. Densmore, R. Passerone, and A. Sangiovanni-Vincentelli. A platform-based taxonomy for ESL design. *IEEE Design and Test of Computers*, 23(5):359– 374, 2006.

[47] R. Dhar, G. George, A. Malani, and P. Steenkiste. Supporting integrated MAC and PHY software development for the USRP SDR. *1st IEEE Workshop on Networking Technologies for Software Defined Radio Networks (SDR)*, 2006.

[48] D. Dietterle. Embedded system protocol design flow based on SDL: from specification to hardware/software implementation. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops (Simutools)*, 2008.

[49] D. Dietterle and R. Kraemer. A hardware accelerated implementation of the IEEE 802.15.3 MAC protocol. *Telecommunication Systems*, 40(3-4):161–167, 2009.

[50] G. Dittmann. *On Instruction-Set Generation for Specialized Processors*. PhD thesis, ETH Zurich, 2005.

[51] C. Doerr, M. Neufeld, J. Filfield, T. Weingart, D. C. Sicker, and D. Grunwald. MultiMAC - an adaptive MAC framework for dynamic radio networking. In *IEEE DySPAN*, 2005.

[52] R. Dömer, A. Gerstlauer, J. Peng, D. Shin, L. Cai, H. Yu, S. Abdi, and D. D. Gajski. System-on-chip environment: a specc-based framework for heterogeneous mpsoc design. *EURASIP Journal of Embedded Systems*, 2008:1–13, 2008.

[53] A. Dominguez, S. Udayakumaran, and R. Barua. Heap data allocation to scratch-pad memory in embedded systems. *Journal of Embedded Computing*, 1(4):521–540, 2005.

[54] S. Edwards, L. Lavagno, E. Lee, and A. Sangiovanni-Vincentelli. Design of embedded systems: Formal models, validation, and synthesis. *Readings in hardware/software co-design*, 2001.

[55] K. Ehrig, C. Ermel, S. Hänsgen, and G. Taentzer. Generation of visual editors as eclipse plug-ins. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering (ASE)*, 2005.

[56] S. Eichler. Performance evaluation of the IEEE 802.11p WAVE communication standard. 2007.

[57] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity – the Ptolemy approach. In *Proceedings of the IEEE*, pages 127–144, 2003.

[58] P. Ezudheen, P. Chandran, J. Chandra, B. P. Simon, and D. Ravi. Parallelizing SystemC kernel for fast hardware simulation on smp machines. 2009.

[59] G. Fankhauser, C. Conrad, E. Zitzler, and B. Plattner. Topsy - a teachable operating system. Technical report, 1997.

[60] W. Fornaciari, P. Gubian, D. Sciuto, and C. Silvano. Power estimation of embedded systems: a hardware/software codesign approach. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 6(2):266 –275, 1998.

[61] M. Frigo, Steven, and G. Johnson. The design and implementation of fftw3. In *Proceedings of the IEEE*, volume 93, pages 216–231, 2005.

[62] T. Fujisawa, J. Hasegawa, K. Tsuchie, T. Shiozawa, T. Fujita, T. Saito, and Y. Unekawa. A single-chip 802.11a MAC/PHY with a 32-b RISC processor. *IEEE Journal on Solid-State Circuits*, 38:2001–2009, 2003.

[63] F. Fummi, G. Perbellini, M. Loghi, and M. Poncino. ISS-centric modular hw/sw co-simulation. In *Proceedings of the 16th ACM Great Lakes symposium on VLSI (GLSVLSI)*, 2006.

[64] L. Gao, S. Kraemer, et al. An integrated performance estimation approach in a hybrid simulation framework. In *Proceedings of the 4th Annual Workshop on Modeling, Benchmarking and Simulation (MoBS)*, 2008.

[65] D. Genbrugge and L. Eeckhout. Chip multiprocessor design space exploration through statistical simulation. *IEEE Transactions on Computers*, 58:1668–1681, 2009.

[66] A. Gordon-Ross, N. Dutt, et al. Automatic tuning of twolevel caches to embedded applications. In *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*, 2004.

[67] M. Gries. Methods for evaluating and covering the design space during early design development. *Integrated VLSI Journal*, 38(2):131–183, 2004.

[68] M. Gries and K. Keutzer. *Building ASIPs: The Mescal Methodology*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[69] M. Gries, C. Kulkarni, C. Sauer, and K. Keutzer. Comparing analytical modeling with simulation for network processors: a case study. In *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*, 2003.

[70] M. Gries, C. Kulkarni, C. Sauer, and K. Keutzer. Exploring trade-offs in performance and programmability of processing element topologies for network processors. In *in Proceedings of Network Processor Workshop in conjunction with Ninth International Symposium on High Performance Computer Architecture (HPCA)*, 2003.

[71] T. Grötker. *System Design with SystemC*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[72] Y. Grunenberger, M. Heusse, F. Rousseau, and A. Duda. Experience with an Implementation of the Idle Sense Wireless Access Method. In *Proceedings of the 2007 ACM CoNEXT conference (CoNEXT)*, 2007.

[73] B. Hailpern and P. Tarr. Model-driven development: the good, the bad, and the ugly. *IBM Systems Journal*, 45(3):451–461, 2006.

[74] M. Handley, E. Kohler, A. Ghosh, O. Hodson, and P. Radoslavov. Designing extensible IP router software. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation (NSDI)*, 2005.

[75] M. Hannikainen, J. Knuutila, T. Hamalainen, and J. Saarinen. Using SDL for implementing a wireless medium access control protocol. In *Proceedings of the 2000 International Conference on Microelectronic Systems Education (MSE)*, 2000.

[76] L. Haque and M. J. Armstrong. A survey of the machine interference problem. *European Journal of Operational Research*, 179(2):469–482, June 2007.

[77] M. Haroud and A. Biere. Hw accelerated ultra wide band MAC protocol using SDL and SystemC. *Proceedings of the IEEE Radio and Wireless Conference (RAWCON)*, 2004.

[78] D. Hedde, P.-H. Horrein, F. Petrot, R. Rolland, and F. Rousseau. A MP-SoC prototyping platform for flexible radio applications. In *Proceedings of the 2009 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools (DSD)*, 2009.

[79] G. R. Hiertz, D. Denteneer, L. Stibor, Y. Zang, X. P. Costa, and B. Walke. The IEEE 802.11 universe. *IEEE Communications Magazine*, 48(1):62–70, January 2010.

[80] J. Hoffman, D. A. Ilitzky, A. Chun, and A. Chapyzhenka. Architecture of the scalable communications core. In *Proceedings of the First International Symposium on Networks-on-Chip (NOCS)*, Washington, DC, USA, 2007.

[81] HomeGrid-Forum. ITU G.hn protocol standard, website. `www.homegridforum.org`, accessed June 2010.

[82] I.-P. Hong, Y.-J. Lee, et al. Multi-threading processor architecture for wireless LAN MAC controller. In *Proceedings of the International Conference on Consumer Electronics (ICCE)*, 2005.

[83] C. H. Hoogendoorn. A general model for memory interference in multiprocessors. *IEEE Transactions on Communication*, (10):998–1005, Oct. 1977.

[84] hostapd. IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS authenticator. `http://hostap.epitest.fi/hostapd/`.

[85] Y. Hwang, G. Schirner, S. Abdi, and D. D. Gajski. Accurate timed rtos model for transaction level modeling. In *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*, 2010.

[86] IEEE 802.11. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specification. *IEEE standard*, 1997.

[87] IEEE 802.11-2007. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications. *IEEE standard*, June 2007.

[88] IEEE 802.11e-2005. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, amendment 8: Medium Access Control (MAC) Quality of Service enhancements. *IEEE standard*, 2005.

[89] IEEE 802.11n/D9.0. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, amendment 5: enhancements for Higher Throughput. *IEEE Unapproved Draft Standard*, 2009.

[90] IEEE 802.11P. Usage models. *IEEE 802.11 standard contributions*, 2004.

[91] M. Iliopoulos and T. Antonakopoulos. Optimised reconfigurable MAC processor architecture. In *8th IEEE ICECS*, 2001.

[92] I. Inan, F. Keceli, and E. Ayanoglu. Modeling the 802.11e enhanced distributed channel access function. In *IEEE Global Telecommunications Conference (GLOBECOM)*, 2007.

[93] Intel Corporation. Intel® Wireless WiFi Link 4965AGN. `http://www.intellinuxwireless.org`, accessed Jan. 2008.

[94] B. L. Jacob, P. M. Chen, S. R. Silverman, and T. N. Mudge. An analytical model for designing memory hierarchies. *IEEE Transactions on Computers*, 45(10):1180–1194, 1996.

[95] A. Jantsch and I. Sander. Models of computation and languages for embedded system design. 2005.

[96] J.-P. Javaudin, M. Bellec, D. Varoutas, and V. Suraci. OMEGA ICT project: Towards convergent gigabit home networks. In *Proceedings of annual the IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2008.

[97] F. K. Jondral. Software-defined radio-basic and evolution to cognitive radio. *EURASIP Journal on Wireless Communication and Networking*, 2005.

[98] A. Jow, C. Schurgers, and D. Palmer. CalRadio: a portable, flexible 802.11 wireless research platform. In *Proceedings of the 1st international workshop on System evaluation for mobile platforms (MobiEval)*, New York, NY, USA, 2007.

[99] G. Kahn. The semantics of a simple language for parallel programming. In J. L. Rosenfeld, editor, *Proceedings of the IFIP Congress on Information Processing*, pages 471–475. North-Holland, New York, NY, 1974.

[100] H. Kalte, M. Porrmann, and U. Rückert. A prototyping platform for dynamically reconfigurable system on chip designs. In *Proceedings of the IEEE Workshop Heterogeneous Reconfigurable Systems on Chip (SoC)*, 2002.

[101] Y. Kanada. Two rule-based building-block architectures for policy-based network control. In *Proceedings of the 2nd International Working Conference on Active Networks Control (IWAN)*, 2000.

[102] T. Kangas, P. Kukkala, H. Orsila, E. Salminen, M. Hännikäinen, T. D. Hämäläinen, J. Riihimäki, and K. Kuusilinna. UML-based multiprocessor SoC design framework. *ACM Transactions on Embedded Computing Systems*, 5(2):281–320, 2006.

[103] K. Karras, D. Llorente, T. Wild, and A. Herkersdorf. Improving memory subsystem performance in network processors with smart packet segmentation. In *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, 2008.

[104] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty. Model-integrated development of embedded software. *Proceedings of the IEEE*, 91:145–164, January 2003.

[105] T. Kempf, M. Doerper, R. Leupers, G. Ascheid, H. Meyr, T. Kogel, and B. Vanthournout. A modular simulation framework for spatial and temporal task mapping onto multi-processor SoC platforms. In *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*, 2005.

[106] T. Kempf, K. Karuri, S. Wallentowitz, G. Ascheid, R. Leupers, and H. Meyr. A sw performance estimation framework for early system-level-design using fine-grained instrumentation. In *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*, 2006.

[107] K. Keutzer, A. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli. System-level design: orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(12):1523–1543, Dec. 2000.

[108] A. Khattab, J. Camp, C. Hunter, P. Murphy, A. Sabharwal, and E. W. Knightly. WARP: a flexible platform for clean-slate wireless medium access protocol design. *SIGMOBILE Mob. Comput. Commun. Rev.*, 12(1):56–58, 2008.

[109] B. Kienhuis, E. Deprettere, et al. An approach for quantitative analysis of application-specific dataflow architectures. In *IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, 1997.

[110] J. P. C. Kleijnen. An overview of the design and analysis of simulation experiments for sensitivity analysis. *European Journal of Operational Research*, 164(2):287–300, 2005.

[111] T. Kogel, A. Wieferink, R. Leupers, G. Ascheid, and H. Meyr. Virtual architecture mapping: A SystemC based methodology for architectural exploration of system-on-chip designs. In *Proceedubgs of the International workshop on Systems, Architectures, Modeling and Simulation (SAMOS*, 2003.

[112] E. Kohler. *The Click modular router*. PhD thesis, Massachusetts Institute of Technology, November 2000.

[113] E. Kohler, R. Morris, B. Chen, et al. The Click modular router. *ACM Transactions on Computer Systems*, 18(3), Aug. 2000.

[114] A. K. Kruth. *The Impact of Technology Scaling on Integrated Analogue CMOS RF Front-Ends for Wireless Applications*. PhD thesis, RWTH, Aachen, 2008.

[115] P. Kukkala, V. Helminen, M. Hannikainen, and T. Hamalainen. Uml 2.0 implementation of an embedded wlan protocol. In *Proceedings of annual the IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2004.

[116] C. Kulkarni, G. Brebner, and G. Schelle. Mapping a domain specific language to a platform FPGA. In *Proceedings of the 41st Design Automation Conference (DAC)*, 2004.

[117] C. Kulkarni, M. Gries, C. Sauer, and K. Keutzer. Programming challenges in network processor deployment. In *Proceedings of the international conference on Compilers, architecture and synthesis for embedded systems (CASES)*, 2003.

[118] S. Künzli, F. Poletti, L. Benini, and L. Thiele. Combining simulation and formal methods for system-level performance analysis. In *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*, 2006.

[119] K. Lahiri, A. Raghunathan, and S. Dey. System-level performance analysis for designing on-chip communication architectures. *IEEE Transactions on Computer Aided-Design of Integrated Circuits and Systems*, 2001.

[120] D. Langen, A. Brinkmann, and U. Ruckert. High level estimation of the area and power consumption of on-chip interconnects. In *Proceedings 13th Annual IEEE International ASIC/SOC Conference*, 2000.

[121] Lantiq. Data sheet, XWAVE 1650. Technical report, 2010.

[122] E. Lee and S. Neuendorffer. Concurrent models of computation for embedded software. *IEEE Proceedings on Computers and Digital Techniques*, 152(2):239–250, Mar 2005.

[123] E. Lee and T. Parks. Dataflow process networks. *IEEE Proceedings*, 83(5):773–801, May 1995.

[124] E. A. Lee. The problem with threads. *Computer*, 39(5):33–42, 2006.

[125] E. A. Lee, X. Liu, and S. Neuendorffer. Classes and inheritance in actor-oriented design. *ACM Transactions on Embedded Computing Systems*, 8(4):1–26, 2009.

[126] H. Lee and T. Mudge. A dual-processor solution for the mac layer of a software defined radio terminal. pages 257–265, 2005.

[127] J. Lee and S. C. Park. Hardware architecture exploration of ieee 802.11n receiver using systemc transaction level modeling. In *Advanced Communication Technology, The 9th International Conference on*, volume 3, pages 1707 – 1710, 2007.

[128] J.-S. Lee, Y.-W. Su, and C.-C. Shen. A comparative study of wireless protocols: Bluetooth, UWB, ZigBee, and Wi-Fi. In *33rd Annual IEEE Conference of the Industrial Electronics Society (IECON)*, 2007.

[129] C. Lefurgy, E. Piccininni, and T. Mudge. Evaluation of a high performance code compression method. In *Proceedings of the 32nd annual ACM/IEEE international symposium on Microarchitecture (MICRO)*, 1999.

[130] H. Lekatsas and W. Wolf. Code compression for embedded systems. In *Proceedings of the 35th annual Design Automation Conference (DAC)*, 1998.

[131] N. Letor, P. De Cleyn, and C. Blondia. Enabling cross layer design: adding the MadWifi extensions to Nsclick. In *Proceedings of the 2nd international conference on Performance evaluation methodologies and tools (ValueTools)*, 2007.

[132] Leupers, R. and Wahlen, O. and Hohenauer, M. and Kogel and others, P. An Executable Intermediate Representation for Retargetable Compilation and High-Level Code Optimization. In *International Workshop on Systems, Architecturs, Modeling and Simulation (SAMOS)*, 2003.

[133] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. Tinyos: An operating system for sensor networks. *Ambient Intelligence*, pages 115–148, 2005.

[134] M. Li, M. Claypool, and R. Kinicki. Playout buffer and rate optimization for streaming over IEEE 802.11 wireless networks. *ACM Transactions on Multimedia Computing Communication Applications*, 5(3):1–25, 2009.

[135] T. Li, Q. Ni, D. Malone, D. Leith, Y. Xiao, and T. Turletti. Aggregation with fragment retransmission for very high-speed WLANs. *IEEE/ACM Transactions on Networks*, 17(2):591–604, 2009.

[136] H. Lichte, S. Valentin, and H. Karl. Automated development of cooperative MAC protocols. *Mobile Networks and Applications*, 2009.

[137] Y.-D. Lin, J.-H. Yeh, T.-H. Yang, C.-Y. Ku, S.-L. Tsao, and Y.-C. Lai. Efficient dynamic frame aggregation in ieee 802.11s mesh networks. *Int. J. Commun. Syst.*, 22(10):1319–1338, 2009.

[138] Y.-D. Lin, J.-H. Yeh, T.-H. Yang, C.-Y. Ku, S.-L. Tsao, and Y.-C. Lai. Efficient dynamic frame aggregation in ieee 802.11s mesh networks. *Int. J. Communication Systems*, 22(10):1319–1338, 2009.

[139] Linux Wireless. mac80211. `http://linuxwireless.org`, accessed Jan. 2008.

[140] D. Liu, A. Nilsson, E. Tell, D. Wu, and J. Eilert. Bridging dream and reality: Programmable baseband processors for software-defined radio. *Communications Magazine, IEEE*, 47(9):134 –140, 2009.

[141] H. Liu and Y. Wang. An efficient software/hardware architecture for the IEEE 802.11n BlockACK mechanism. In *Proceedings of the 11th IEEE Singapore International Conference on Communication Systems (ICCS)*, pages 1111–1114, Nov. 2008.

[142] H. P. Loeb, R. Buchty, and W. Karl. A network agent for diagnosis and analysis of real-time ethernet networks. In *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems (CASES)*, 2006.

[143] M. Loghi, F. Angiolini, D. Bertozzi, L. Benini, and R. Zafalon. Analyzing on-chip communication in a mpsoc environment. In *Proceedings of the conference on Design, automation and test in Europe (DATE)*, 2004.

[144] M.-H. Lu, P. Steenkiste, and T. Chen. Using commodity hardware platform to develop and evaluate csma protocols. In *Proceedings of the third ACM international workshop on Wireless network testbeds, experimental evaluation and characterization (WiNTECH)*. ACM, 2008.

[145] S. Malik, M. Martonosi, and Y.-T. S. Li. Static timing analysis of embedded software. In *Proceedings of the 34th annual Design Automation Conference (DAC)*, pages 147–152, New York, NY, USA, 1997.

[146] A. Mann, B. Grube, I. Konorov, S. Paul, L. Schmitt, D. McElroy, and S. Ziegler. A sampling ADC data acquisition system for positron emission tomography. *IEEE Transactions on Nuclear Science*, 53(1):297 –303, 2006.

[147] A. Martin, P. Starr, and P. Larson. Software requirements for interventional MR in restorative and functional neurosurgery. *Neurosurgery clinics of North America*, 20(2):179–86, 2009.

[148] G. Martin. Overview of the MPSoC design challenge. In *Proceedings of the 43rd Annual Conference on Design Automation (DAC)*, 2006.

[149] Marvell. 88W87 integrated MAC/Baseband/RF SoC product brief. http://www.marvell.com/products/wireless/8786.pdf, 2009.

[150] M. Masmano, I. Ripoll, P. Balbastre, and A. Crespo. A constant-time dynamic storage allocator for real-time systems. *Real-Time Systems*, 40(2):149–179, 2008.

[151] H. Massalin. Superoptimizer: a look at the smallest program. *SIGPLAN Notes*, 22(10):122–126, 1987.

[152] K. Masselos and N. S. Voros. Implementation of wireless communications systems on FPGA-based platforms. *EURASIP Journal on Embedded Systems*, 2007(1):1–1, 2007.

[153] R. McIlroy, P. Dickman, and J. Sventek. Efficient dynamic heap allocation of scratch-pad memory. In *Proceedings of the 7th international symposium on Memory management (ISMM)*, 2008.

[154] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *SIGCOMM Computer Communication Review*, 38:69–74, March 2008.

[155] S. Mehta, D. Weber, et al. An 802.11g WLAN SoC. *IEEE Journal of Solid-State Circuits*, 40(12):2483 – 2491, 2005.

[156] T. Meyer, P. Langend¨orfer, V. Suraci, S. Nowak, and M. Bahr. An Inter-MAC architecture for heterogeneous gigabit home networks. In *Proceedings of the IEEE Conference on Local Computer Networks (LCN)*, 2009.

[157] T. Meyerowitz, A. Sangiovanni-Vincentelli, M. Sauermann, and D. Langen. Source-level timing annotation and simulation for a heterogeneous multiprocessor. In *Proceedings of the conference on Design, automation and test in Europe (DATE)*, 2008.

[158] A. Mihal and K. Keutzer. A processing element and programming methodology for click elements. In *Proceedings of the Workshop on Application Specific Processors (WASP 2005)*, 2005.

[159] A. C. Mihal. *Deploying Concurrent Applications on Heterogeneous Multiprocessors*. PhD thesis, EECS Department, University of California, Berkeley, November 10 2006.

[160] MIPS Technologies. Homepage. `www.mips.com`, accessed 2009.

[161] MIPS Technologies. MIPS SDE 6.x programmers' guide. http://www.mips.com, accessed 2008.

[162] MIPS Technologies. M4K core product overview. http://www.mips.com, accessed 2009.

[163] J. Mitola. The software radio architecture. *Communications Magazine, IEEE*, 33(5):26–38, 1995.

[164] T. N. Mudge and H. B. Al-Sadoun. A semi-markov model for the performance of multiple-bus systems. *IEEE Transactions on Computers*, 34(10):934–942, 1985.

[165] J. Mudigonda, H. M. Vin, and R. Yavatkar. Overcoming the memory wall in packet processing: Hammers or ladders? In *Proceedings of the 2005 Symposium on Architecture for Networking and Communications Systems (ANCS)*, 2005.

[166] S. Nabi, C. Wells, and W. Vanderbauwhede. A dynamically reconfigurable hardware co-processor for a multi-standard wireless MAC Processor. *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2008.

[167] Y. Nagai, A. Fujimura, Y. Shirokura, Y. Isota, F. Ishizu, H. Nakase, S. Kameda, H. Oguma, and K. Tsubouchi. 324mbps WLAN equipment with MAC frame aggregation. In *Proceedings of the IEEE 17th International Symposium on Personal, Indoor and Mobile Radio Communications*, 2006.

[168] S. Neuendorffer and E. A. Lee. Hierarchical reconfiguration of dataflow models. In *MEMOCODE*, pages 179–188, 2004.

[169] M. Neufeld, J. Fifield, C. Doerr, A. Sheth, and D. Grunwald. SoftMAC - flexible wireless research platform. In *Workshop on Hot Topics in Networks (HotNets)*, November 2005.

[170] M. Neufeld, A. Jain, and D. Grunwald. Nsclick: bridging network simulation and deployment. In *Proceedings of the 5th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems (MSWiM)*, 2002.

[171] J.-C. Niemann, C. Puttmann, M. Porrmann, and U. Rückert. GigaNetIC - a scalable embedded on-chip multiprocessor architecture for network applications. In *Workshop proceedings of the 19th International Conference on Architecture of Computing Systems (ARCS)*, 2006.

[172] G. Nychis, T. Hottelier, Z. Yang, S. Seshan, and P. Steenkiste. Enabling MAC protocol implementations on software-defined radios. In *Proceedings of the 6th USENIX symposium on Networked systems design and implementation (NSDI)*, 2009.

[173] R. Ohlendorf, A. Herkersdorf, and T. Wild. FlexPath NP: a network processor concept with application-driven flexible processing paths. In *Proceedings of the 3rd IEEE/ACM/IFIP international conference on hardware/software codesign and system synthesis (CODES+ISSS)*, 2005.

[174] V. Oksman and S. Galli. G.hn: the new ITU-T home networking standard. *Communications Magazin*, 47(10):138–145, 2009.

[175] S. Ortiz. IEEE 802.11n: The road ahead. *Computer*, 42(7):13–15, 2009.

[176] O. Ozturk, M. Kandemir, M. Irwin, and S. Tosun. Multi-level on-chip memory hierarchy design for embedded chip multiprocessors. In *Proceedings of the 12th International Conference on Parallel and Distributed Systems (ICPADS)*, 2006.

[177] P. R. Panda, F. Catthoor, et al. Data and memory optimization techniques for embedded systems. *Transactions on Design Automation of Electronic Systems (TODAES)*, 2001.

[178] P. R. Panda, S. Inc, N. D. Dutt, R. Nicolau, F. Catthoor, A. V, E. Brockmeyer, C. Kulkarni, and E. D. Greef. Large embedded memories. *IEEE Design & Test of Computers*, 2008.

[179] G. Panic, D. Dietterle, et al. A system-on-chip implementation of the IEEE 802.11a MAC layer. In *Proceedings of The Euromicro Conference on Digital System Design (DSD)*, 2003.

[180] S. Pasricha, N. Dutt, and M. Ben-Romdhane. Extending the transaction level modeling approach for fast communication architecture exploration. In *Proceedings of the 41st annual conference on Design automation (DAC)*, 2004.

[181] D. A. Patterson and J. L. Hennessy. *Computer architecture: a quantitative approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[182] J. M. Paul, D. E. Thomas, and A. S. Cassidy. High-level modeling and simulation of single-chip programmable heterogeneous multiprocessors. *ACM Transactions on Design Automations for Electronic Systems*, 10(3):431–461, 2005.

[183] P. Paulin, C. Pilkington, and E. Bensoudane. StepNP: A system-level exploration platform for network processors. *IEEE Design & Test of Computers*, 19(6):17–26, Nov.–Dec. 2002.

[184] E. Perahia and R. Stacey. *Next Generation Wireless LANs: Throughput, Robustness, and Reliability in 802.11n*. Cambridge University Press, 2008.

[185] P. Petrus, Q. Sun, S. Ng, J. Cho, et al. An integrated draft 802.11n compliant MIMO baseband and MAC processor. *Digest of Technical Papers, IEEE International Solid-State Circuits Conference (ISSCC)*, 2007.

[186] A. D. Pimentel, M. Thompson, S. Polstra, and C. Erbas. Calibration of abstract performance models for system-level design space exploration. *J. Signal Process. Syst.*, 50:99–114, February 2008.

[187] W. L. Plishker. Automated mapping of domain specific languages to application specific multiprocessors. Technical Report UCB/EECS-2006-123, 2006.

[188] M. Porrmann, J. Hagemeyer, J. Romoth, and M. Strugholtz. Rapid prototyping of next-generation multiprocessor SoCs. In *Semiconductor Conference Dresden (SCD)*, 2009.

[189] A. Pullini, F. Angiolini, P. Meloni, D. Atienza, S. Murali, L. Raffo, G. De Micheli, and L. Benini. NoC Design and Implementation in 65 nm Technology. In *Proceedings of the First International Symposium on Networks-on-Chip (NOCS), year = 2007, unit = LSI ESL*.

[190] Qi Chen et al. Overhaul of IEEE 802.11 modeling and simulation architecture in NS-2. In *Proceedings of the 10th International Symposium on Modeling, Analysis, and Simulation of Wireless and Mobile Systems (MSWIM)*, 2007.

[191] Ralink Corp. RT2880 MIMO Wireless Access Point/Router/iNIC SoC. Product Sheet, accessed online, 2010.

[192] U. Ramacher. Software-defined radio prospects for multistandard mobile phones. *Computer*, 40(10):62–69, 2007.

[193] A. Rasche and A. Poize. Dynamic reconfiguration of component-based real-time software. In *Proceedings of the IEEE Workshop on Object-Oriented Real-Time Dependable Systems (WORDS)*, 2005.

[194] J. Regehr. *Handbook of Real-Time and Embedded Systems*, chapter Safe and structured use of interrupts in real-time and embedded software. CRC Press, 2006.

[195] J. Regehr and N. Cooprider. Interrupt verification via thread verification. *Electronic Notes on Theoretical Computer Science*, 174(9):139–150, 2007.

[196] S. Rein, C. Gühmann, and F. Fitzek. Compression of short text on embedded systems. *Journal of Computers (JCP)*, 2006.

[197] M. Reiser and S. S. Lavenberg. Mean-value analysis of closed multichain queuing networks. *Journal of the ACM (JACM)*, 27(2):313–322, 1980.

[198] M. Reshadi, P. Mishra, and N. Dutt. Hybrid-compiled simulation: An efficient technique for instruction-set architecture simulation. *ACM Trans. Embed. Comput. Syst.*, 8(3):1–27, 2009.

[199] K. Richter, M. Jersak, and R. Ernst. A formal approach to MpSoC performance verification. *Computer*, 36(4):60–67, 2003.

[200] J. Rowson. Hardware/software co-simulation. In *Proceedings of the 31st Design Automation Conference (DAC)*, 1994.

[201] E. Rubow, R. McGeer, J. C. Mogul, and A. Vahdat. Chimpp: A click-based programming and simulation environment for reconfigurable networking hardware. Technical report, HP Labs, HPL-2010-25, 2010.

[202] A. Salhotra, R. Narasimhan, and R. Kopikare. Evaluation of contention free bursting in IEEE 802.11e wireless LANs. *Proceedings of the IEEE Wireless Communications and Networking Conference*, 2005.

[203] S. Samadi, A. Golomohammadi, A. Jannesari, et al. A novel implementation of the IEEE 802.11 medium access control. In *Proceedings of the International Symposium on Intelligent Signal Processing and Communications (ISPACS)*, 2006.

[204] A. Sangiovanni-Vincentelli. Quo vadis, SLD? Reasoning about the trends and challenges of system level design. *Proceedings of the IEEE*, 95(3):467–506, 2007.

[205] A. Sangiovanni-Vincentelli and G. Martin. Platform-based design and software design methodology for embedded systems. *IEEE Design & Test*, 18(6):23–33, 2001.

[206] S. Sankaran, M. Zargari, L. Nathawad, et al. Design and implementation of a CMOS 802.11n SoC), journal = IEEE Communications Magazine, year = 2009, volume = 47, pages = 134-143, number = 4, month = April , timestamp = 2009.10.18.

[207] C. Sauer. *Application-driven development of flexible packet-oriented communication interfaces*. Ph.d. thesis, University of California at Berkeley, Berkeley, CA, 2009.

[208] C. Sauer, M. Gries, and S. Dirk. Hard- and software modularity of the NOVA MPSoC platform. In *Proceedings of the conference on Design, automation and test in Europe (DATE)*, 2007.

[209] C. Sauer, M. Gries, J.-C. Niemann, M. Porrmann, and M. Thies. Application-Driven Development of Concurrent Packet Processing Platforms. In *Proceedings of the International Symposium on Parallel Computing in Electrical Engineering (PARELEC)*, 2006.

[210] C. Sauer, M. Gries, and S. Sonntag. Modular domain-specific implementation and exploration framework for embedded SW platforms. In *Proceedings of the Design Automation Conference (DAC)*, 2005.

[211] G. Schirner, A. Gerstlauer, and R. Domer. Automatic generation of hardware dependent software for mpsocs from abstract system specifications. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC 2008)*, 2008.

[212] G. Schirner, A. Gerstlauer, and R. Dömer. Fast and accurate processor models for efficient MPSoC design. *ACM Transactions on Design Automation for Electronic Systems*, 15(2):1–26, 2010.

[213] S. Schliecker, A. Hamann, R. Racu, and R. Ernst. Formal methods for system level performance analysis and optimization. In *In Proceedings of the Design Verification Conference*, 2008.

[214] J. Schnerr, O. Bringmann, A. Viehl, and W. Rosenstiel. High-performance timing simulation of embedded software. In *Proceedings of the 45th annual conference on Design automation (DAC)*, 2008.

[215] J. Shafer and S. Rixner. A gigabit reconfigurable programmable network interface card. Annual Affiliates Meeting, Department of Electrical and Computer Engineering, Rice University, 2005.

[216] N. Shah. Understanding network processors. Master's thesis, UC Berkeley, 2001.

[217] N. Shah, W. Plishker, K. Ravindran, and K. Keutzer. NP-Click: a productive software development approach for network processors. *IEEE Micro*, 24(5):45–54, Sept.–Oct. 2004.

[218] M. Shalan and V. J. Mooney, III. Hardware support for real-time embedded multiprocessor system-on-a-chip memory management. In *Proceedings of the 10th international symposium on Hardware/software codesign (CODES)*, 2002.

[219] A. Sharma and E. M. Belding. FreeMAC: framework for multi-channel MAC development on 802.11 hardware. 2008.

[220] T. Shono, Y. Shirato, H. Shiba, et al. IEEE 802.11 wireless LAN implemented on software defined radio with hybrid programmable architecture. *IEEE Transactions on Wireless Communications*, 4(5):2299–2308, 2005.

[221] D. Skordoulis, Q. Ni, H.-H. Chen, A. Stephens, C. Liu, and A. Jamalipour. IEEE 802.11n MAC frame aggregation mechanisms for next-generation high-throughput WLANs. *Wireless Communications, IEEE*, 15(1):40–47, February 2008.

[222] Y. Song, J. yon Choi, Y. Kim, H. Park, and S. kyu Lee. MAC implementation for IMT-advanced multi-gigabit nomadic systems. *Proceedings of the IEEE 19th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2008.

[223] S. Sonntag, M. Gries, and C. Sauer. SystemQ: Bridging the gap between queuing-based performance evaluation and systemc. *Design Automation for Embedded Systems*, 11(2):91–117, 2007.

[224] C. Soviani, I. Hadžić, and S. A. Edwards. Synthesis of high-performance packet processing pipelines. In *Proceedings of the 43rd annual Design Automation Conference (DAC)*, 2006.

[225] M. Stege. A flexible prototyping platform for wireless communication systems. In *Wireless Word Research Forum (WWRF 12)*, 2004.

[226] M. Streubühr, J. Gladigau, C. Haubelt, and J. Teich. Efficient Approximately-Timed Performance Modeling for Architectural Exploration of MPSoCs. In *Proceedings of the Forum on specification and Design Languages (FDL)*, 2009.

[227] V. Suhendra, C. Raghavan, and T. Mitra. Integrated scratchpad memory optimization and task scheduling for mpsoc architectures. In *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems (CASES)*, 2006.

[228] J. Sztrik. Finite source queuing systems and their applications: a bibliography. Technical report, Institute of Mathematics and Informatics, University of Debrecen, Debrecen, Hungary, 2001.

[229] K. Tan, J. Zhang, J. Fang, et al. Sora: high performance software radio using general purpose multi-core processors. 2009.

[230] The Click Modular Router Project. http://pdos.csail.mit.edu/click/, accessed March 2009.

[231] M. Tsai, C. Kulkarni, C. Sauer, N. Shah, and K. Keutzer. A benchmarking methodology for network processors. In *Network Processor Design: Issues and Practices*, pages 141–165. Morgan Kaufmann Publishers, 2002.

[232] S. Valentin, H. S. Lichte, H. Karl, et al. Cooperative wireless networking beyond store-and-forward. *Wireless Personal Communications*, 48(1):49–68, 2009.

[233] J. R. Van Houten, J. P. Jarosz, B. J. Welch, D. E. Gallegos, and M. W. Learn. Soft-core processor study for node-based architectures. Technical report, Sandia National Laboratories, TRN: US200902-166, 2008.

[234] L. VanderPerre, B. Bougard, J. Craninckx, et al. Architectures and circuits for software defined radios: scaling and scalability for low cost and low energy. In *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC)*, 2007.

[235] A. Varga. The OMNeT++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference*, 2001.

[236] M. Verma, S. Steinke, and P. Marwedel. Data partitioning for maximal scratchpad usage. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC 2008)*, 2003.

[237] V. Visoottiviseth, T. Piroonsith, and S. Siwamogsatham. An empirical study on achievable throughputs of ieee 802.11n devices. In *Proceedings of the 7th international conference on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOPT)*, 2009.

[238] C.-Y. Wang and H.-Y. Wei. IEEE 802.11n MAC enhancement and performance evaluation. *Mobile Networks and Applications (Journal)*, 14(6):760 – 771, 2009.

[239] Z. Wang, W. Haberl, A. Herkersdorf, and M. Wechs. Syscola: A framework for co-development of automotive software and system platform. In *Proceedings of the 46th Design Automation Conference (DAC)*, 2009.

[240] Z. Wang and A. Herkersdorf. An efficient approach for system-level timing simulation of compiler-optimized embedded software. In *Proceedings of the 46th Annual Design Automation Conference (DAC)*.

[241] D. Wendlandt, M. Casado, P. Tarjan, and N. McKeown. The clack graphical router: visualizing network software. In *Proceedings of the 2006 ACM symposium on Software visualization (Soft-Vis)*, 2006.

[242] B. Wheeler and L. Gwennap, editors. *A Guide to Network Processors.* The Linley Group, 7th edition, 2005.

[243] A. Wieferink, T. Kogel, R. Leupers, et al. A system level processor/communication co-exploration methodology for multi-processor system-on-chip platforms. In *Proceedings of the conference on Design, automation and test in Europe (DATE)*, 2004.

[244] S. Wiethölter and C. Hoene. Design and verification of an IEEE 802.11e EDCF simulation model in ns-2.26. Technical Report TKN-03-019, Telecommunication Networks Group, Technische Universität Berlin, 2003.

[245] Wipro Newlogic. WiLD 802.11 a/b/g MAC, product sheet. http://www.newlogic.com/products/802_11_wireless_abg/, accessed 2010.

[246] W. Wolf. *High-Performance Embedded Computing: Architectures, Applications, and Methodologies*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[247] H. Wu, Y. Liu, Q. Zhang, and Z.-L. Zhang. SoftMAC: Layer 2.5 collaborative MAC for multimedia support in multihop wireless networks. *IEEE Transactions on Mobile Computing*, 6(1):12–25, 2007.

[248] Z.-H. Yang, Y.-T. Chen, and C.-P. Fan. Design and verification of high-throughput IEEE 802.11 MAC-layer hardware IP with FPGA platform. In *Journal of Chinese Institute of Engineer (JCIE)*, 2010.

[249] S.-R. Yoon, J. Lee, and S.-C. Park. Design space exploration of ieee 802.11n using systemc. pages 19–23, 2007.

[250] H. Zabel and W. Müller. Increased accuracy through noise injection in abstract RTOS simulation. In *Proceedings of the conference on Design, automation and test in Europe (DATE)*, 2009.

[251] J. Zambreno, D. Nguyen, and A. Choudhary. Exploring area/delay tradeoffs in an AES FPGA implementation. In *In Proceedings of the 14th Annual International Conference on Field-Programmable Logic and Applications (FPL*, 2004.

[252] Y. Zhao. A model of computation with push and pull processing. Technical Report UCB/ERL M03/51, EECS Department, University of California, Berkeley, 2003.

Gedruckt auf alterungsbeständigem Papier nach ISO 9706