An FPGA-Accelerated Testbed for Hardware Component Development in MIMO Wireless Communication Systems

Filippo Borlenghi, Dominik Auras, Ernst Martin Witte, Torsten Kempf,
Gerd Ascheid, Rainer Leupers, Heinrich Meyr
Institute for Communication Technologies and Embedded Systems
RWTH Aachen University, 52056 Aachen, Germany
email: {borlenghi,auras,witte,kempf,ascheid,leupers,meyr}@ice.rwth-aachen.de

Abstract—**FPGA-based prototyping is nowadays common practice in the functional verification of hardware components since it allows to cover a large number of test cases in a shorter time compared to HDL simulation. In addition, an FPGA-based emulator significantly accelerates the simulation with respect to bit-true software models. This speed-up is crucial when the statistical properties of a system have to be analyzed by Monte Carlo techniques. In this paper we consider a multiple-input multiple-output (MIMO) wireless communication system and show how integrating an FPGA accelerator in the software simulation framework is key to enable the development of complex hardware components in the receiver, from algorithm all the way to chip testing. In particular, we focus on a MIMO detector implementation based on the depth-first sphere decoding algorithm. The speed-up of up to 3 orders of magnitude achieved by hardware-accelerated simulation compared to a pure software testbed enables an extensive fixed-point exploration. Furthermore, it allows a unique characterization of the system communication performance and the MIMO detector run-time characteristics, which vary for different configuration parameters and operating scenarios and hence require a thorough investigation.**

I. Introduction

Nowadays FPGAs are widely used not only as target devices for hardware implementation but also as integral part of the development process, acting as emulators and simulation accelerators. Simulation speed is a relevant issue for complex systems with multiple operational modes and configurations since in such cases a slow simulator may prevent the coverage of a sufficient number of test cases in the verification phase. The domain of wireless communications provides good examples of such systems. In fact, the physical layer of modern standards, such as IEEE 802.11n and LTE Advanced, employs advanced transmission techniques, such as spatial-multiplexing multiple-input multiple-output (MIMO) schemes, and supports many different modes to adapt to different scenarios and target applications. A mode is defined, among others, by the modulation scheme, the number of transmit and receive antennas and the forward error-correction (FEC) code. In the case of IEEE 802.11n, the combination of these parameters results in more than 30 different modes [1], thus introducing relevant challenges for system design and verification.

This work has been supported by the Ultra High-Speed Mobile Information and Communication (UMIC) Research Centre, RWTH Aachen University.

Furthermore, the complete characterization of a communication system requires the measurement of statistical properties (e.g., the transmission error rate). Typically, these characteristics cannot be derived analytically and have to be extracted from extensive Monte Carlo simulations. In addition, they have to be tracked throughout the development process to ensure that algorithmic transformations and fixed-point optimizations do not degrade the system performance beyond an acceptable margin. The characterization process does not necessarily end after finalizing the hardware implementation. In fact, a modern wireless receiver has to cope with the variety of modes defined by communication standards and with different environmental conditions, described by different channel models. Moreover, the system typically features several configuration options, for instance to tune at run-time the trade-off between communication performance and complexity/efficiency. Verifying the system behavior in all possible conditions becomes quickly infeasible and therefore a subset of relevant test cases has to be defined to achieve a sufficient verification confidence level in an acceptable time. However, characterizing the statistical properties of the system can be of interest for additional scenarios and typically requires much longer simulations than pure verification purposes. The case study on MIMO detection implementation in Sec. IV shows how the proper characterization of a real component requires an extremely large number of simulations. In such cases, simulation speed is a limiting factor which can be overcome by hardware emulation.

A. Algorithm-to-Hardware Design Flow

The development of a signal-processing hardware component, as described for instance in [2], typically starts from the specification of the algorithm in a high-level language, such as Matlab or C++, using floating-point operations. As noted in [3], two major steps lead from the initial algorithmic specification to the hardware implementation:

1. *Fixed-point conversion*: all arithmetic operations are replaced by their equivalent fixed-point representation and subsequently suitable word widths for all input, output and

intermediate values are determined. Since unnecessarily wide word widths result in a significant hardware overhead, they have to be minimized for the specified acceptable performance loss. In most cases word widths cannot be derived analytically and have to be iteratively refined based on the performance loss measured by Monte Carlo simulations.

2. *Architecture design*: the data and control paths of the hardware architecture are defined by the designer based on the algorithm and on the target requirements in terms of area, throughput, latency and energy consumption. The architecture is then formally specified on register-transfer level (RTL) in a hardware description language (e.g., Verilog or VHDL) or on a higher-level in an architecture description language [4] when the implementation targets a programmable processing element; in the latter case, the RTL model is automatically generated from the high-level specification.

Once the RTL model is available, the development enters a third phase, where the hardware specification has to be verified against the corresponding fixed-point bit-true software model. Verification requires again extensive simulations to ensure the correct behavior of the hardware in all the relevant test cases.

While this design flow does not necessarily require FPGA-based emulation, this can become crucial in several steps if the simulator is not fast enough. In such a case, the design phases can be interleaved to fully exploit the accelerator, as shown in Fig. 1. First, a conservative choice of fixed-point word widths is made ensuring that no performance loss with respect to the floating-point reference is introduced; this setup is then used to implement a first version of the RTL model. After fixing the first and typically most severe bugs with HDL simulation, the model can be synthesized on the FPGA and integrated in the simulation environment. At this point, the acceleration provided by the emulator can be exploited both for more extensive verification and for fine tuning the fixed-point word widths. Since the performance of a communication system is assessed by global measures, a key requirement is the tight integration of the hardware accelerator in the overall simulation environment to enable the observation of the global behavior throughout the development of the single components. Sec. IV shows how the described flow has been applied to the design of an ASIC for MIMO detection, taped out in 90 nm CMOS technology and successfully tested [5].

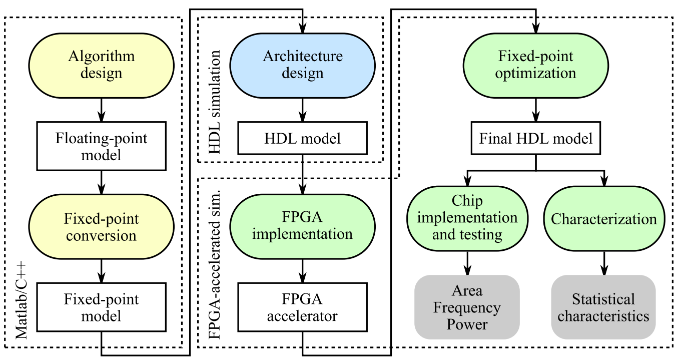

Fig. 1. Algorithm-to-hardware design flow.

B. Related Work

The focus of this work is on building an environment for efficient and consistent algorithm-to-hardware development and characterization of MIMO wireless receiver components, where FPGA-based emulation plays a key role. This approach is similar to [3], where however FPGA prototyping is employed for the whole transceiver baseband processing rather than on a finer-grained component level as in this work. Besides enabling to identify component-specific effects on the global system behavior, the possibility of developing single processing elements independently helps when different designers focus on different receiver parts.

Unlike our approach, other works target FPGA-based emulation mainly for prototyping and demonstrating a complete system, possibly running in real time. Testbeds such as those presented in [6]–[11] are extremely valuable to demonstrate the potential of a new technology and gain experience in real-world issues, such as RF impairments and particular channel conditions. In such cases, hardware emulation is rather the outcome of the development than an integral part of it.

In other cases, such as [12], the FPGA implementation is automatically generated from a high-level Matlab specification for speeding up the algorithmic exploration with the minimum hardware implementation effort. This approach restricts the implementation target to FPGAs; moreover, the efficiency resulting from high-level synthesis approaches is typically not sufficient in terms of throughput and energy consumption to cope with the requirements of modern standards and hence manual optimizations by the designer are still needed. In this paper we describe a flow conceived to facilitate the hardware designer's task rather than provide an automatic path from algorithm to hardware.

II. MIMO System Simulation Framework

Spatial-multiplexing MIMO transmission is a key technology for increasing data rates of present and future wireless communication standards. As shown in Fig. 2, MIMO schemes transmit multiple data streams in parallel over M_T antennas to increase spectral efficiency, ideally by a factor of M_T. The receiver, hereby assumed to have $M_\mathrm{R} = M_\mathrm{T}$ antennas, first cancels the inter-antenna interference introduced by the MIMO channel, represented by the channel matrix $\mathbf{H} \in \mathbb{C}^{M_\mathrm{R} \times M_\mathrm{T}}$, and detects the transmitted symbol vector $\mathbf{s} = [s_1, ..., s_{M_\mathrm{T}}]^T \in \mathcal{O}^{M_\mathrm{T}}$ from the received symbol vector $\mathbf{y} \in \mathbb{C}^{M_\mathrm{R}}$, with $\mathcal{O}$ being the complex QAM modulation alphabet and each scalar symbol $s_i \in \mathcal{O}$ maps to Q bits. The detected symbol vector is then demapped into the corresponding vector of $M_\mathrm{T}Q$ bits, which is finally passed to the channel decoder for error correction by means of convolutional, turbo or LDPC codes among others. Since the main focus of this work is on baseband digital signal processing in the receiver, the system model assumes that the impairments and non-idealities introduced by the RF and analog frontend are modeled as part of the channel.

A key step in the recovery of the transmitted signal is the *detection* of the symbol vector $\mathbf{s}$, whose complexity greatly

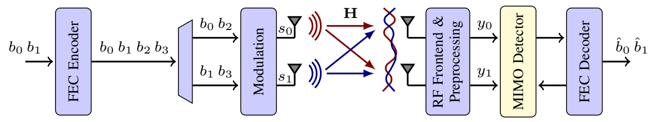

Fig. 2. Block diagram of a MIMO system with $M_{\mathrm{T}} = M_{\mathrm{R}} = 2$.

increases in the MIMO case compared to a single-antenna system. This task can be performed in different ways depending on the target complexity-performance trade-off: while *hard-output* MIMO detectors [13], [14] only provide a binary estimate of the bits thus limiting the overall performance of the system, *soft-output* detectors [15] compute the reliability of each bit. This soft information can be exploited by the channel decoder to enhance significantly its error-correcting capability. Furthermore, according to the principle of bit-interleaved coded modulation with iterative decoding (BICM-ID) [16], a feedback loop can be introduced to iteratively exchange information between MIMO detector and channel decoder, thus improving the overall communication performance. Due to the large performance gains of this iterative detection and decoding (IDD) scheme and the corresponding increase in the MIMO detector complexity, this topic is of high scientific interest. Only recently the first hardware implementations of iterative MIMO detectors have been presented in [17] and [18]; the latter is subject of the case study in Sec. IV.

From a simulation perspective, the BICM-ID principle defines the structure of the receiver and the information which is exchanged between components, i.e., their interfaces, as shown in Fig. 2. On the contrary, the inner algorithm of each component is not specified right from the start, either because there are multiple options available (e.g., the channel decoding algorithm depends on the applied FEC code) or because the algorithmic exploration is part of the development. Based on these observations, a simulator has been developed in Matlab/Simulink to facilitate the algorithmic development. While the structure of the receiver and the interfaces between components are precisely defined, the inner implementation of each component can be easily exchanged. This concept allows in a first phase the evaluation of different algorithms for the same task. Once the algorithm is finalized, its implementation on different levels of abstraction can be used inside the component, including all the required simulation models:

1. *Floating-point model*: written in Matlab, it provides the starting point and reference for the implementation.

2. *Fixed-point model*: also written in Matlab by using an efficient fixed-point library developed in house, it represents the golden reference for hardware verification. If white-box verification is targeted, the fixed-point model has to mimic the hardware behavior since intermediate values as well as inputs and outputs have to be computed in the same way as in the architecture, resulting in a slower simulation and less readable code. Although white-box verification greatly helps debugging at an early stage, a black-box input/output testing style is preferrable to speed up later verification phases.

3. *Architecture model*: depending on the architectural choice, the hardware model can run on an instruction-set simulator in the case of a programmable processing element or an HDL simulator (e.g., Mentor Graphics Modelsim) for RTL code. These tools can be connected to Matlab using inter-process communication (IPC) mechanisms (e.g., UNIX pipes) for on-line co-simulation; in this case, the Matlab side of the component under test implements the interface with the hardware simulator rather than the actual processing. Alternatively, the communication can be implemented off-line by dumping from Matlab reference values which are later read by the hardware simulator.

4. *FPGA-based emulator*: similar to the IPC connection used for hardware simulators, the Matlab block implements the communication and synchronization with the software layer controlling the FPGA board; Sec. III describes the corresponding implementation details.

Integrating and exchanging the different models in a unified simulation environment is essential to ensure consistency throughout the development process and especially in the verification phase, as stated in [3]. Furthermore, this approach is particularly suitable to achieve reproducible simulation results, which is another key requirement for verification. These properties enable to use the same testbed to generate vectors for testing and characterizing the silicon implementation after fabrication. The selection of the inner implementation of each component, as well as the related settings (e.g., run-time parameters and fixed-point word widths) and the global simulation setup (e.g., number of antennas, modulation scheme, channel model, signal-to-noise ratio, etc.), is specified in a single configuration file, which is used to initialize the simulator. This solution avoids scattering the many available settings across different locations and further facilitates consistency.

Although the Matlab environment allows an easy algorithmic development, its main drawback is the low simulation speed, especially when the modeling style gets closer to a hardware-like implementation, as required for bit-true models. Highly-optimized software models are a viable option to speed up the simulation, at the cost of additional development effort. FPGA-based emulation provides an alternative which improves not only the simulation speed but also the efficiency of the hardware verification process.

III. FPGA-Based Emulator Integration

Applications from the communication domain have particular characteristics which result in specific requirements for hardware emulation. First of all, emulation is typically used to verify and prototype the complete functionality of a system. However, in the domain of wireless communications the emulated component may be only a part of a complex system simulation, where the non-emulated parts still need high computational power. Since this work does not target live demonstration, there are no real-time requirements and hence computationally-intensive tasks are not timing-critical, although they should not represent a bottleneck for the overall simulation. Moreover, a communication system is characterized by its statistical properties, such as error rates, which are determined by extensive Monte Carlo simulations. Applying

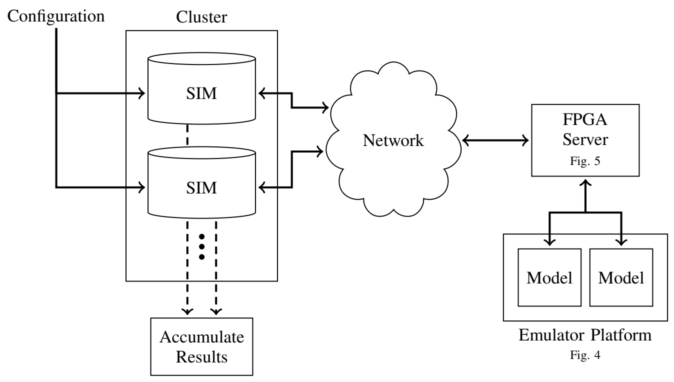

Fig. 3. Hardware-accelerated simulation infrastructure.

the Monte Carlo technique allows to partition the simulations into several independent runs, which can be executed in parallel with an additional final result aggregation pass.

By observing these specific properties and requirements, a generic concept for integrating hardware emulators in a communication system simulation is elaborated. The optimization target is the maximization of the total simulation throughput, i.e., the number of simulated transmitted bits per second, in order to obtain statistical data with a higher confidence level in the same time or shorter simulation times for the same amount of data, thus ultimately shortening the design cycle.

As shown in Fig. 3, the testbed revolves around a central FPGA-based emulator acting as a server which collects data from the different simulations running on a Linux-based parallel computing cluster, feeds them to the FPGA and finally redistributes the results. This scalable approach allows to achieve a high utilization of the emulator, which is an expensive resource and hence should be used in the most efficient way, and to easily share its access among multiple users. Moreover, executing the non-accelerated computationally-intensive tasks in parallel on a multitude of cheap PC hosts avoids the creation of bottlenecks for the utilization of the emulator and the overall simulation throughput. In fact, typically a single simulation is unable to fully load the FPGA accelerator. Hence several data streams from different simulations are aggregated by the server to increase the load and consequently the utilization. Furthermore, the stream property of the transceiver simulation, where each component executes only after the previous one has completed its processing, simplifies interaction and synchronization between the emulator and the software part and hence also the aggregation of concurrent simulations.

Thanks to the scalability of this approach together with the Monte Carlo property of the simulations, the achievable concurrency degree is only limited on the one hand by the available computing cluster resources and on the other hand by the maximum load sustainable by the emulator. In general, as shown in Sec. IV, it is preferrable to operate in a condition of slight overloading of the emulator in order to ensure its 100 % utilization at any time. Since there are no real-time requirements, the additional waiting time to get access to the emulator in overloading conditions is not an issue. For the same reason, no special and expensive low-latency network is needed and a standard Ethernet infrastructure can be used.

The proposed solution for integrating the emulator in the software testbed consists essentially of three parts: the FPGA hardware protoype, the FPGA management server and the simulator client side. The following sections describe in details the relevant implementation aspects of each component.

A. *FPGA-Based Hardware Emulator*

The goal of this work is to provide a quick prototyping solution for testing, verifying and characterizing a hardware design which is later implemented on a standard-cell library. Therefore, the hardware architecture is not modified to exploit FPGA-specific optimizations, such as deep pipelining or the explicit usage of specialized FPGA resources (e.g., embedded multipliers or other custom DSP blocks), which could improve the resource utilization and the timing of the FPGA implementation. Accordingly, the clock frequency of the emulator is typically set to a conservative target value. Similarly, the interface of the hardware component has to be preserved. In order to achieve this goal, the model under test is integrated in a wrapper which implements protocol adaptation between the stream-based software simulation and the HDL model interface. In this way, the FPGA-specific communication issues are abstracted from the hardware component and its model does not have to be modified to be synthesized on different targets; only if its interface changes the protocol adapter has to be modified accordingly. This approach is essential to efficiently exploit the FPGA acceleration in early design-space exploration phases, when the model is not finalized yet.

The FPGA-based emulator used in this work connects to the host computer via a peripheral component interface (PCI). As shown in Fig. 4, FIFOs are inserted between the PCI endpoint and the protocol adapter to buffer the I/O streams. This additional buffering increases the overall throughput by ensuring that the emulator is not stalled by either the lack of input data or a full output buffer. Moreover, FIFOs seamlessly adapt the I/O rate between the emulated model and the host, enabling different clock domains for the transfer logic and the model itself. Typically, the PCI endpoint is clocked at the host interface frequency of 33 MHz, while the emulated model may run on a different and possibly independent clock. From the host point of view, the FPGA is accessed as memory-mapped

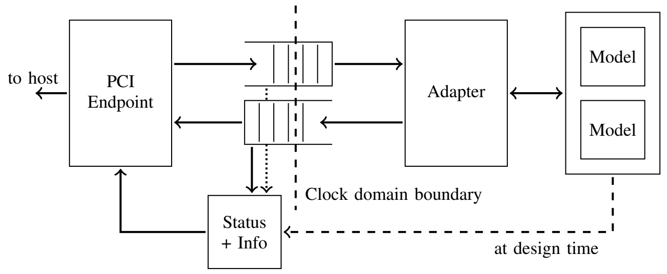

Fig. 4. FPGA interface design.

I/O. When multiple instances of the emulated model run in parallel on the FPGA, their address spaces are separated to enable the management server to access them independently. Any scheduling and data reordering operation required by the multiple cores is implemented in server software; in fact, a hardware implementation of these tasks would increase the design effort with limited benefits, since the delay introduced by these operations is not critical for non-real-time simulation.

Besides implementing data processing, the FPGA prototype provides additional information, such as the design version currently running, the available configuration parameters and their current settings. In this way, the FPGA management server and the simulation clients can check for consistency between the requested configuration at the client side and the emulated one, thus ensuring the validity of the results.

B. FPGA Management Server

The main purpose of the server software is to manage the simulation data streams to and from the FPGA device. As shown in Fig. 5, this task includes serializing the concurrent independent input streams, scheduling and dispatching them to the FPGA device and finally collecting and returning the results to the connected clients. To this end, a TCP server listens for incoming requests and, after a protocol initialization phase, for every accepted connection opens a new socket for data communication, managed by a corresponding thread. Since the input streams are completely independent the server synchronizes them by queueing them in a buffer, protected from data hazards with a software mutex. The main server thread pulls data from the input queue, then schedules, assigns and dispatches the input to the FPGA device via the PCI bus. This task is particularly critical for efficiently exploiting the multiple design instances emulated on the FPGA. Upon the completion of the computation, the results are transferred asynchronously back to the data sockets, possibly in a different order than the input data was accepted.

The specific client and server implementations are totally independent of each other as long as they comply to a common communication protocol, built on top of TCP/IP. Furthermore, since the aggregation of the concurrent incoming streams and the management of the emulation resources are under the control of the server, the number of concurrent simulations and the number of emulated models are fully decoupled and the clients do not need to be aware of this information. In the common case that the simulation clients outnumber the emulated cores, the server queues the exceeding requests and the clients simply experience a longer waiting time.

C. Simulator Client

On the client side, the simulation includes a model-dependent wrapper which replaces the emulated component. This wrapper collects the input data into network packets to be submitted to the server and translates between simulation and model interfaces, converting for instance the floating-point data into the fixed-point format required by the emulated model and vice versa. The packet transmission is blocking, meaning that the client waits for completion. This approach is devised to keep the inner implementation of the single component independent from its interface, so that a change in the implementation does not affect the rest of the system.

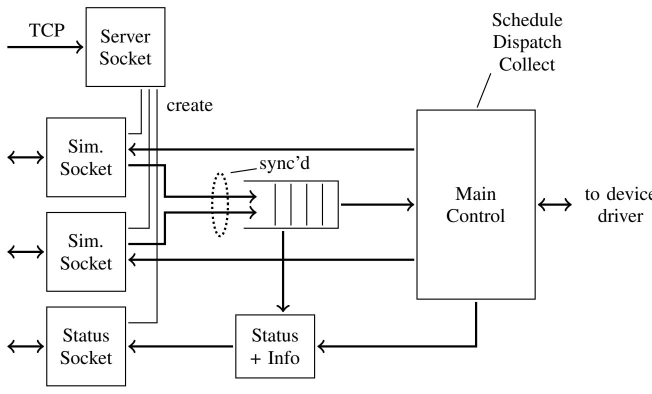

Fig. 5. FPGA management server design.

When the connection with the FPGA server is established, an initialization phase allows the client to retrieve information about the currently emulated model and check that it matches the desired one. This test is important to ensure consistency when different models are emulated concurrently or when the model is under verification and hence new versions with bug fixes are exchanged often. Moreover, the reproducibility of simulation results requires the knowledge of the code and model versions used to produce the data. Since the simulation does not have control on the remote emulated models, the client has to check for consistency and fail at any mismatch.

As mentioned previously, Monte Carlo simulations are parallelized so that a single configuration is split across concurrent runs, each using different random seeds. Once all the runs are finished, their results are aggregated and used to obtain statistical measurements. Partial results, however, are written back already while the simulation is running, thus enabling to monitor the progress and discover problems early.

IV. Case Study: MIMO Detector Implementation and Characterization

Detection is one of the most complex tasks carried out in a MIMO wireless receiver, in particular when IDD is used and the detector has to take into account the additional feedback from the channel decoder. Prominent approaches to iterative MIMO detection are (quasi-)linear algorithms, such as MMSE parallel interference cancellation (PIC) [17], and tree-search schemes, such as single tree-search sphere decoding (STS SD) [19]. While MMSE PIC has a fixed complexity, STS SD achieves a communication performance closer to channel capacity and its complexity, exponential in the worst case, scales with the target performance and the operating conditions. Hence, STS SD is an interesting candidate for implementation.

A major characteristic of STS SD is that detection is performed as a depth-first tree search to minimize redundant calculations. From the implementation standpoint, however, this strategy results in a sequential traversal of the tree.

Moreover, selecting the next tree node to check is a complex task involving heavy arithmetic operations and the (partial) sorting of a subset of nodes (for more details the reader is referred to [19] and [18]). As a consequence, the efficient implementation of STS SD is challenging in hardware but especially difficult in software, since the control-flow overhead associated with the sequential depth-first traversal is even more expensive in software than in hardware.

A. Simulation Setup and Operating Points

In order to compare the simulation speed of different models, it is necessary to first define the setup in use. We consider a MIMO BICM-ID system with 4×4 antennas and a 64-QAM modulation scheme, operating in an i.i.d. Rayleigh fading channel; an LDPC error-correcting code compliant to the IEEE 802.11n standard [1] is used, in a configuration with code-word length equal to 648 information bits and rate $1/2$. The receiver is assumed to have perfect channel knowledge and employs soft-input soft-output STS SD (with sorted QR decomposition pre-processing [20]) and an LDPC decoder based on the layered offset-min-sum algorithm [21] running 10 iterations. The signal-to-noise ratio (SNR) is defined as $\text{SNR} = M_\text{T} E_\text{s}/N_0$, with $E_\text{s} = \mathbb{E}[|s|^2]$, $s \in \mathcal{O}$.

Since the complexity of STS SD varies heavily with the SNR, in the following the analysis is restricted to two meaningful operating points. First of all, when optimizing the fixed-point word widths the error rate of the system has to be checked against the best performance achievable with floating-point operations, i.e., in the lowest SNR region where the system can operate. This region corresponds to the highest complexity for STS SD; moreover, in this case run-time constraints cannot be applied to STS SD to avoid introducing an additional performance-penalty factor which would hinder the fixed-point analysis. In the aforementioned setup with a target packet[1] error rate (PER) of 1 % the lowest operating point with 4 detector-decoder iterations corresponds to an SNR of 17.2 dB. Simulation speed is particularly relevant in this scenario since every fixed-point setup has to be verified in this SNR region.

However, one of the goals of the design flow is to enable a full system characterization, which requires simulations over a large SNR range to determine both the error-rate behavior and the statistics of the STS SD variable run-time. Therefore, as a second comparison point we consider a high SNR of 25 dB, where the system can reach the target PER of 1 % with a single pass through detector and decoder and a run-time constraint of M_T nodes on STS SD, corresponding to its minimum complexity and hence to the maximum speed both for the simulation and the hardware implementation.

B. MIMO Detector FPGA Implementation

The emulation platform used in this work is a DINI Group DN6000k10S-2vp100-6 FPGA board (Fig. 6), which features a Xilinx VirtexII-Pro FPGA and connects to a host computer via PCI, as described in Sec. III-A. Up to 4 parallel SD cores (using the final fixed-point word widths) can be mapped on this FPGA, with a resource occupation of 24614 flip-flops (27 % of the total), 80820 4-input look-up tables (91 %), 64 blocks of 18 kB distributed RAM (14 %) and 152 18×18-bit multipliers (34 %). The clock frequency is conservatively set to 11 MHz so that the HDL model can be implemented on the emulation platform without FPGA-specific optimizations.

[1]A packet is assumed to be equal to a code word, i.e., 1296 coded bits

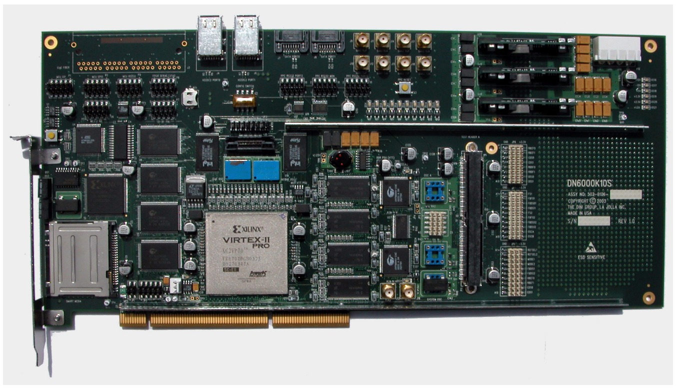
Fig. 6. Target FPGA platform [22].

The I/O interface of the MIMO detector follows a simple scheme. First, all the input data is provided in parallel in the same clock cycle and buffered internally so that it is available throughout the computation. Then, the SD execution is started with a simple handshake signal and its end is signaled again by a single-bit flag. At this point, the output data is stored in an internal buffer which can be read out entirely in parallel. The FPGA protocol adapter which wraps the MIMO detector takes care of translating this interface to the PCI bus.

C. Simulation Speed Comparison

The simulation speed of the different models has been measured in terms of throughput in kbit/s at the receiver output when simulating the two operating points described in Sec. IV-A on a computing cluster. The computational resources of the cluster are heterogeneous in terms of CPU cores and available memory, thus introducing a small uncertainty in the measurement of the simulation speed. However, this uncertainty is reduced by averaging the measurements over a long simulation time and it does not change the conclusions drawn from the following comparison.

Besides the simulation speed of a single instance of the testbed, the scaling of the aggregate throughput of the different models with the number of used CPU cores, i.e., the number of simulation instances running concurrently, is particularly relevant. As shown in Fig. 7 (logarithmic scale) and 8 (linear scale), the speed of the pure software floating- and fixed-point simulations increases linearly with the number of CPU cores since the concurrent instances are completely independent of each other. The speed penalty caused by fixed-point operations ranges from a factor 2 in high SNR to an order of magnitude in low SNR. While this slowdown is expected due to the overhead of emulating in software fixed-point operations, its variability is due to the algorithmic transformations introduced

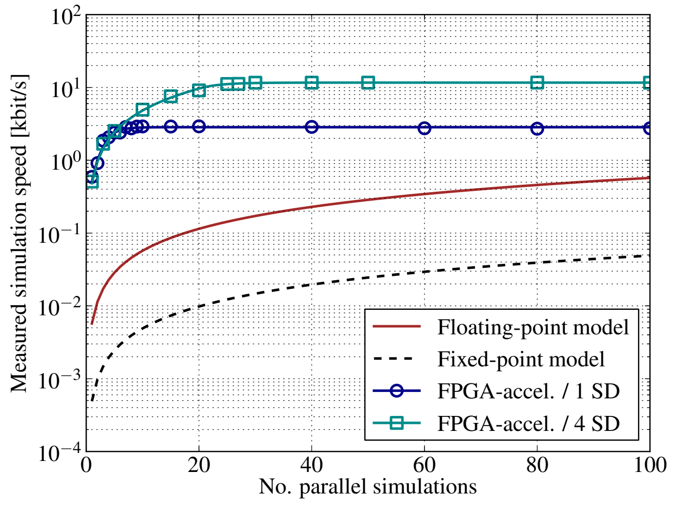

Fig. 7. Simulation speed comparison in low SNR (17.2 dB).

in the fixed-point model to enable an efficient hardware implementation [18]. These modifications do not affect the communication performance of the MIMO detector but its run-time varies with respect to the floating-point model, which implements the algorithm in [19].

As shown in Fig. 7, in low SNR FPGA-accelerated simulations show a different behavior with respect to the parallelism degree. In fact, the multiple simulation instances compete for accessing the emulator as a shared resource and hence the aggregate throughput stops increasing above a certain parallelism degree, corresponding to a fully-loaded FPGA. This saturation occurs at around 8 and 30 parallel simulations respectively for a single- and a quad-core SD FPGA accelerator. The maximum throughput scales linearly with the number of SD cores on the FPGA (2.9 kbit/s for one SD core vs. 11.6 kbit/s for 4 SD cores), showing that this is the limiting factor rather than the interface between the FPGA and the simulator.

With respect to the pure software simulations, FPGA-based emulation enables a huge speed-up, up to 2 and 3 orders of magnitude respectively compared to the floating- and fixed-point models. In order to achieve the speed of the FPGA-accelerated simulation with 4 SD cores and 30 CPU cores, a floating-point simulation would require more than 2000 CPU cores and a fixed-point one more than 23000. In practice, we assume that a measure of the PER is statistically relevant if at least 100 erroneous packets have been received, meaning that for the PER to be reliable down to 10^{-3} at least 10^5 packets (i.e., roughly 65×10^6 uncoded bits) have to be simulated. Therefore, assuming a cluster with 100 CPU cores, verifying a fixed-point setup in software in a single SNR point would require more than *two weeks*, thus hindering a proper fixed-point exploration; FPGA acceleration can reduce the time for the same task to one hour and a half.

In the high-SNR operating point, on the other hand, the complexity of the MIMO detector is at its minimum, thus allowing software models to reduce the gap with respect to the quad-core FPGA-accelerated simulation to a factor of 1.6 and 3.5 respectively for the floating- and fixed-point models, as shown in Fig. 8. However, in this scenario the throughput curve of the FPGA-accelerated testbed does not show saturation effects in the range of parallelism that could be reliably tested in this work (up to 135 CPU cores) and hence its advantage over pure software simulations does not decrease when more CPU cores are available. Another difference in the high-SNR behavior with respect to the low-SNR case is that increasing the number of SD cores on the FPGA does not provide any relevant speed-up anymore. This statement is however only valid in the limited range of parallelism achievable by the computing cluster used in this work; in fact, since the curves in Fig. 8 do not hit the saturation region, the acceleration potential of the FPGA emulator is not fully exploited. For higher numbers of CPU cores, it can be expected that the observations made in the low-SNR case still apply.

D. Enhancing the Development Process

The first step of the design flow which greatly benefits from FPGA emulation is the fixed-point exploration. Since each fixed-point setup has to be verified at the limit of the communication performance, i.e., in low SNR, only FPGA acceleration can enable an extensive analysis, cutting the required time from weeks down to hours. Therefore, in the MIMO detector development the HDL model of the architecture is first defined with design-time configurable fixed-point word widths and then thoroughly verified against the fixed-point software model using conservatively long word widths. Afterwards, this model is synthesized on the FPGA and the word widths are reduced step by step until the communication performance loss, measured by FPGA-accelerated simulations, is within the acceptable range. The HDL model with the final word widths is then verified one last time against the corresponding fixed-point software model. For the chip implementation in [5], this process was repeated for each of the 3 SD cores on the chip, each one supporting a different maximum modulation order. This level of optimization significantly improved area efficiency and would have been impossible without integrating the FPGA-based emulator in the flow.

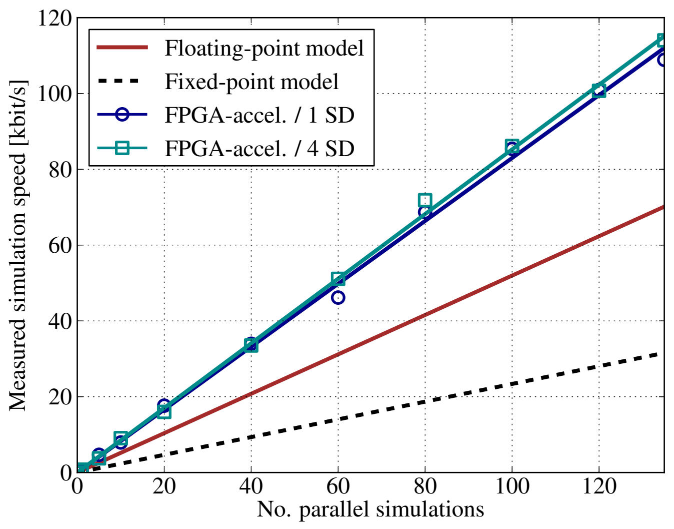

Fig. 8. Simulation speed comparison in high SNR (25 dB).

The verification phase is the next development step that can take advantage of the FPGA prototype. While HDL simulation is necessary to discover and fix the first major bugs, its speed, which is comparable to the Matlab fixed-point model, is insufficient to cover a large number of test cases. Therefore, extensive black-box verification is performed with the help of the emulator; whenever a mismatch in the output results or in the run-time of the MIMO detector with respect to the expected values is encountered, the corresponding test case is isolated and simulated on the HDL simulator to enable white-box verification. This solution enables the coverage of a much higher number of configurations and operating conditions in the verification than stand-alone HDL simulation.

Furthermore, the same facilities used to generate the I/O verification traces can be exploited to produce the test vectors for post-fabrication chip testing. These vectors are typically obtained from HDL simulation, since they have to be cycle-accurate. The FPGA prototype can replace the simulation in this task, leading to significant time savings for generating large amounts of vectors (e.g., the MIMO detector chip in [5] was tested with more than 10^5 symbol vectors).

Finally, verification typically covers a limited subset of relevant test cases and a limited number of vectors per test case. However, the full characterization of the MIMO detector properties requires statistically-relevant simulations of many operating scenarios (defined among others by the SNR, the number of antennas and the modulation scheme) and receiver setups (defined for instance by the run-time constraints on the MIMO detector, the type of channel decoder and the number of iterations in the IDD system). This parameter space quickly grows to thousands of possible combinations and can be covered only by the FPGA-accelerated simulation.

V. Conclusions

The development of modern wireless receiver baseband processing components is challenging in the design of the algorithm and of a corresponding efficient hardware architecture as well as in the complexity of the design process itself. In this paper, we have shown how a conventional design flow can fail to provide the necessary capabilities for developing, verifying and characterizing complex signal processing components. These limitations can be overcome by integrating in the flow a hardware accelerator achieving a sufficient simulation speed to deal with the aforementioned tasks, without compromising the usability of the simulation framework. The proposed approach proved to be essential in enabling the first silicon implementation of soft-input soft-output STS SD presented in [5], from the initial hardware development stages all the way to chip testing and characterization. Due to the independence of the simulation framework from the inner implementation of the single components, this approach is not limited to the presented case study but can be applied similarly to other processing elements of the transceiver or another system with analogous characteristics.

References

[1] IEEE Standards Association, *802.11n-2009 IEEE Standard for Information Technology – Telecommunications and Information Exchange between Systems – Local and Metropolitan Area Networks – Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Higher Throughput*, Std. IEEE 802.11n, Sep. 2009.

[2] J.-W. Weijers *et al.*, “From MIMO-OFDM algorithms to a real-time wireless prototype: A systematic Matlab-to-hardware design flow,” *EURASIP J. Advances in Signal Process.*, vol. 2006, 2006.

[3] P. Greisen, S. Haene, and A. Burg, “Simulation and emulation of MIMO wireless baseband transceivers,” *EURASIP J. Wireless Commun. and Networking*, vol. 2010, Apr. 2010.

[4] P. Mishra and N. Dutt, “Architecture description languages for programmable embedded systems,” *Inst. Electr. Eng. Proc. – Comput. and Digital Techniques*, vol. 152, no. 3, pp. 285–297, May 2005.

[5] F. Borlenghi *et al.*, “A 772 Mbit/s 8.81 bit/nJ 90 nm CMOS soft-input soft-output sphere decoder,” in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, Jeju, Korea, Nov. 2011, pp. 297–300.

[6] S. Haene, D. Perels, and A. Burg, “A real-time 4-stream MIMO-OFDM transceiver: System design, FPGA implementation, and characterization,” *IEEE J. Sel. Areas Commun.*, vol. 26, no. 6, pp. 877–889, Aug. 2008.

[7] S. Caban *et al.*, “Vienna MIMO testbed,” *EURASIP J. Appl. Signal Process.*, vol. 2006, pp. 142–142, Jan. 2006.

[8] J. W. Wallace, B. D. Jeffs, and M. A. Jensen, “A real-time multiple antenna element testbed for MIMO algorithm development and assessment,” in *Proc. IEEE Int. Symp. Antennas and Propagation Soc.*, vol. 2, Jun. 2004, pp. 1716–1719.

[9] P. Murphy *et al.*, “An FPGA based rapid prototyping platform for MIMO systems,” in *Proc. Asilomar Conf. Signals, Syst. and Comput. (ACSSC)*, vol. 1, Nov. 2003, pp. 900–904.

[10] M. Wouters *et al.*, “Real time prototyping of broadband wireless LAN systems,” in *Proc. IEEE Int. Workshop Rapid Syst. Prototyping*, Jun. 2004, pp. 226–231.

[11] W. Zhu, D. Browne, and M. Fitz, “An open access wideband multiantenna wireless testbed with remote control capability,” in *Proc. Int. Conf. Testbeds and Research Infrastructures for the Develop. of Networks and Communities (Tridentcom)*, Feb. 2005, pp. 72–81.

[12] L. G. Barbero and J. S. Thompson, “Rapid prototyping system for the evaluation of MIMO receive algorithms,” in *Proc. Int. Conf. Comput. as a Tool (EUROCON)*, vol. 2, Nov. 2005, pp. 1779–1782.

[13] A. Burg *et al.*, “VLSI implementation of MIMO detection using the sphere decoding algorithm,” *IEEE J. Solid-State Circuits*, vol. 40, no. 7, pp. 1566–1577, Jul. 2005.

[14] Z. Guo and P. Nilsson, “Algorithm and implementation of the K-best sphere decoding for MIMO detection,” *IEEE J. Sel. Areas Commun.*, vol. 24, no. 3, pp. 491–503, Mar. 2006.

[15] C. Studer, A. Burg, and H. Bölcskei, “Soft-output sphere decoding: Algorithms and VLSI implementation,” *IEEE J. Sel. Areas Commun.*, vol. 26, no. 2, pp. 290–300, Feb. 2008.

[16] X. Li and J. A. Ritcey, “Bit-interleaved coded modulation with iterative decoding using soft feedback,” *IET Electron. Lett.*, vol. 34, no. 10, pp. 942–943, May 1998.

[17] C. Studer, S. Fateh, and D. Seethaler, “A 757 Mbit/s 1.5 mm^2 90 nm CMOS soft-input soft-output MIMO detector for IEEE 802.11n,” in *Proc. European Solid-State Circuits Conf. (ESSCIRC)*, Seville, Spain, Sep. 2010, pp. 530–533.

[18] E. M. Witte *et al.*, “A scalable VLSI architecture for soft-input soft-output single tree-search sphere decoding,” *IEEE Trans. Circuits Syst. II*, vol. 57, no. 9, pp. 706–710, Sep. 2010.

[19] C. Studer and H. Bölcskei, “Soft-input soft-output single tree-search sphere decoding,” *IEEE Trans. Inf. Theory*, vol. 56, no. 10, pp. 4827–4842, Oct. 2010.

[20] D. Wübben *et al.*, “Efficient algorithm for decoding layered space-time codes,” *IET Electron. Lett.*, vol. 37, no. 22, pp. 1348–1350, Oct. 2001.

[21] C. Roth *et al.*, “A 15.8 pJ/bit/iter quasi-cyclic LDPC decoder for IEEE 802.11n in 90 nm CMOS,” in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, Beijing, China, Nov. 2010, pp. 1–4.

[22] DINI Group DN6000k10S VirtexII-Pro based ASIC prototyping engine. http://dinigroup.com/DN6000k10s.php