Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT

Signal Processing 83 (2003) 1427–1444

SIGNAL PROCESSING

www.elsevier.com/locate/sigpro

Rapid prototyping for wireless designs: the five-ones approach

Markus Rupp[a,*,1], Andreas Burg[b,2], Eric Beck[c]

[a] *TU Wien, Gusshausstr 25/389, Vienna 1040, Austria*
[b] *ETH-Zurich, Integrated Systems Laboratory, Gloriastr. 35, CH-8092 Zurich, Switzerland*
[c] *Bell-Labs, Lucent Technologies, Wireless Research Laboratory, 791 Holmdel-Keyport Road, Holmdel, NJ 07733-0400, USA*

Abstract

In a highly innovative market, wireless systems nowadays undergo very short production cycles. Due to these tough timing constraints, the time-consuming process of prototyping is often neglected, jeopardizing the entire product becoming successful. Heavy application of automatic tools can allow for rapid prototyping overcoming this unfortunate situation and de-risking the product challenge. However, the application of automatic tools alone does not speed up the prototyping process sufficiently. By reflecting on current design processes, several paradigms for faster prototyping are concluded, named the *Five-Ones Approach*: One team, One environment, One code, One documentation and One code revision tool. Based on such a Five-Ones Approach, a consistent prototyping environment to implement a prototyping design from first idea to final implementation is presented in this paper. In particular, the design of a prototype for a MIMO system with four transmit and four receive antennas, based on the current UMTS FDD downlink standard is reported.

Keywords: Wireless MIMO systems for UMTS; Rapid prototyping; Code refinement; Electronic design automation; Five-ones approach

1. Introduction

Consider a so-called *high-technology development environment* defined by a high rate of innovation from one product to the next. Wireless communication is currently such a high-technology field in which the classical development methods do not satisfy the needs of a quick turnaround of ideas into products [31]. In classic product designs, prototypes are built beforehand to study their behaviour under *real-world* conditions and to ensure that there are no technological flaws in the design that could prevent the product from realization. They also have the advantage that they can be presented to potential customers long before the product is ready and convince them at an early time of the advantages the future product will provide. However, due to the severe time constraints in the wireless market, prototypes are often skipped and the design team relies only on simulation results and the experience with previous products. Due to their high complexity, modern wireless systems are often heterogeneous systems consisting of many commercial DSPs, FPGA and specific ASIC solutions. If the effort to build a prototype for such a system is roughly as much as that of building the whole product, the

* Corresponding author. Tel.: +431-58801-38967.
E-mail addresses: mrupp@nt.tuwien.ac.at (M. Rupp), apburg@iis.ee.ethz.ch (A. Burg), ericbeck@lucent.com (E. Beck).
[1] This work has in part been supported by the Christian Doppler Laboratory for Design Methodology of Signal Processing Algorithms at the TU Wien, Austria.
[2] This work has in part been supported by the Bell-Laboratories, NJ.

0165-1684/03/$ - see front matter © 2003 Elsevier Science B.V. All rights reserved.
doi:10.1016/S0165-1684(03)00090-2

question is why build a prototype at all since it would hardly be available before the product. Further arguments to abandon time-consuming real-time experiments are their *small flexibility* preventing exploration of the whole parameter space and the *lack of perfect control* of the test environment as it is common in simulations, making it very hard to compare the prototype behaviour with simulation results and thus to draw conclusions.

Under the pressure to be first on the market, telecommunication equipment companies often decide to focus more on simulations and trust the expertise of their research team. Such a development style can easily lead to failure, of which the development of HiperLAN/1 is a good example.[3] In the course of its development, many companies were involved in the standardization process, typically represented by members of their research teams. It was believed that a standard equalizer would solve all transmission problems caused by inter-symbol-interference (ISI) of such wireless channels. Although equalizer algorithms are well explored in literature and showed very promising behaviour in simulations, once the equalizer was to be implemented, its realization turned out to be unexpectedly difficult. Particular solutions had to be investigated [5,40,47] in order to meet the requirements (the solutions in the given references only allow equalization for wireless channels up to 50 ns RMS, certainly small office spaces only) causing a delay in the production process and an unexpected increase in realization costs. Today (March 2003), almost 7 years after the termination of the standardization process, only one company has a product on the market [21] and others are not expected. Simulation alone, with its perfect double-precision numerical environment, clearly gave the wrong impression that the equalizer would be a simple task to implement. By now, one should have learnt that not all that works in MATLAB, works in the real world (as well). The impact on the wireless field is not always that dramatic. However, being late on the market [4] due to problems occurring in the last phases of the design cycle is very costly and has to be prevented.

Recognizing these problems, companies have started their own initiatives to overcome the situation. Infineon, for example, introduced the idea of *virtual prototyping* in their design process [7,17], a concept in which a complete software model of the later hardware components is specified and implemented at an early stage. It can be used as a reference model at later design stages and much more importantly allows the software development team to work early with a model that behaves identically to the future product. Such a method, however, also has its shortcomings. With the high complexity of UMTS systems, for example, it will hardly be possible to perform extensive simulations with a model containing all technical details. Functional test vectors, for example, are known to increase by a factor of 100 every 6 years [17]. Also, and maybe more importantly, the physical world of the (more or less) unknown radio channels is only represented by simplified and therefore less precise mathematical models. If, for example, the behaviour of a channel estimation algorithm at 500 kmph needs to be investigated in a rich scattering environment, no reliable models are available. Particularly in the wireless field, this point attracts much attention since the physical channels defining the transmission rate of future systems are typically unknown and only given in form of statistical models (see for example [19,23]). It is thus important to come up with a *rapid prototyping* of such technical features that are new and carry much risk. If they are not understood well enough, the product realization is in jeopardy.

Unfortunately, building a prototype for something as complex as a base station with all its features would require almost the same effort as building the product itself. Rapid prototyping has thus undergone some paradigm change. It used to be a starter equipment to evaluate a concept as well as to show customers the companies' potential in coming out with new products. Nowadays rapid prototyping has come to mean something different: it is not the entire, but only a partial set of features of the final product that is to be built in a rapid, but sometimes (hardware) inefficient way. Hence, only such technical parts that are new to the realization team [10,15,34] are implemented. Therefore, rapid prototyping does not require all steps of production and can be available much earlier. For example, marketing strategies, form factors, power consumption, standard compliance and cost are usually no or only minor issues for the development of a prototype. Only its technical behavior is important. The

[3] Another example is the explosion of an early Ariane rocket due to numerical errors, though not for the wireless market [13].

open questions to tackle are feasibility, complexity and the behaviour of the new product in a real rather than a simulated environment. Nevertheless, with the ever increasing complexity of modern wireless products, rapid prototyping requires fairly complex and heterogeneous hardware set-ups that cannot be handled by a single person. For example, UMTS transceivers functionality cannot be realized by one or two powerful DSPs due to the required bit-level operations on very high speed (see for example the announcements of TIs C64 processors [3,29,43]). A rapid prototyping team is thus needed (typically of the size 10–20 people) to cope with the various technologies required in this field in order to build a complete radio link for the UMTS physical layers.

On the other hand, rapid prototyping is not very different from a product design since it requires a team with almost the same set of skills. The only difference is that fewer people are required and clearly their skill sets must have a rich repertoire of many aspects in electronic design. In the following Section 2, the classical product development flow will be reviewed to understand the different functions of the various teams involved. From this study, it can be learnt what slows the development of wireless designs and thus in which way improvement is possible, to accomplish prototyping in a rapid fashion. Many of such improvements are not only applicable for speeding up prototyping, but also the production process. However, in this paper the focus is entirely on the rapid prototyping design flow. Furthermore, Section 2.4 will present a short overview of existing EDA tools as they are currently available on the market, their advantages and shortcomings and bring them into their relation to the requirements of the various design teams. [4]

Section 3 will draw first conclusions from the reflections and analyses presented in the second section and propose a new design approach called the Five-Ones Approach. Section 4, will report on a case study of a wireless multiple-input–multiple-output (MIMO) design for the UMTS FDD downlink with four transmit and four receive antennas that was finished recently. A complete real-time wireless transceiver based on TI-C6x DSPs and Xilinx V1000 FPGAs is presented. Along with the various design steps, our particular realization of the proposed Five-Ones Approach will be described.

2. Classical product development in wireless: a review

The following three teams can be distinguished: research team, design specification team, and implementation team. Particular aspects to production as outlook design, compatibility to existing products and standards, marketing, production testing, and distribution are neglected.

2.1. The research team

In a classical design process, a research team (also called forward-looking or pre-development team) works out new technological ideas to include new features or improve existing ones. The *verification* of such new ideas is typically performed by simulations. Favoured are MATLAB from MATHWORKS [37] or simply C-programming. Note that graphical tools like SIMULINK from MATHWORKS, COSSAP/CoCentric System Studio from SYNOPSYS [11] or Signal Processing Workstation (SPW) from CADENCE [38] are rarely used in this context since the researchers feel restricted by the formalism and constraints of such tools. [5] Such constraints will be discussed in more detail in Section 3 and solutions presented in Section 4. In order to focus on new technological ideas of the design, not all details are included in the simulation and the transmission chain is not described in full detail. Only the most important entities describing such parts that are essential for the transmission of data are modelled and many details are therefore left out when they are not considered to have a substantial impact on the final product behaviour. Such simulations are often performed assuming base-band signals due to the limited performance

[4] Note that the focus here is on commercially available tools, rather than university tools because commercial tools are well documented and maintained and their cost is rarely an issue for a prototype. Nevertheless, university tools are often the source for very inspiring concepts.

[5] There are many similar but less known tools available: Khoros, RTExpress, GEDAE, RIPPEN, Talaris to name a few.

of workstations and PCs well knowing that parts of the design are running in a higher frequency domain. Such high-frequency parts are typically neglected entirely, or if necessary, for showing recent improvements, are transformed into the base-band domain. Note that typically *only results are reported* in publications and internal technical reports. Little attention is devoted to the description of the simulation program itself and the included details.

2.2. The system design team

Once the simulation results of the research team are satisfactory, a second so-called design specification, system design or system architecture team comes into play. Their first interaction is rather passive, as they are being taught the newly developed methods from the first team. The publications and technical reports of the research group, often not suited for the average design engineer are a further support for the system design team.

The design team now decides what techniques will be used to map such algorithmic ideas into software and hardware, in particular, which components are mapped into general purpose processors and which require dedicated hardware architectures (partitioning). They define what specifics these components need to satisfy and what development and final product costs are anticipated. All components and interfaces between such components need to be specified. Standard components that are available and satisfy all conditions to support the newest product idea are usually bought, while previously built functions, so-called intellectual property (IP), are reused wherever possible. It is the responsibility of this group to select available components at the lowest cost and to make the product fit into the already existing product line. Such effort can change a product entirely. While giving the product a first shape in terms of general system architecture, little or no architectural optimization at the detail level is made here.

During the course of the design team's work, many new questions arise about the feasibility of the future product. These questions are fed back to the research team and typically answered by including more details into the simulations. Having the design teams modifying the code themselves typically fails due to a lack of program description. Note also that by defining HW and SW blocks, this team already defines a HW/SW split. This split may be coarse, but can have considerable impact later in the design process. Remarkably, such partitioning is nowadays typically performed based on the team's experience without any tool support.

2.3. The product realization team

Once the system has been specified in detail, a third group of engineers, the implementation team or realization team becomes involved. Their task is to build all required components that cannot be bought from outside vendors and to join them with existing parts. In short: they have to make the product work. The implementation team has knowledge of the specific tools that are required to program DSPs, FPGAs, and to design ASICs. They need to be taught the innovations of the first team, supplemented by the implementation guidelines of the design team. Note that the work of the implementation team is very labour intense and it is very important for its timely success that the first two groups had no flaws left in their designs. A major re-design in the middle of the building process can be as costly as starting anew.

The algorithmic implementation requires a permanent co-simulation of the actual implementation code with the original high-level simulation. If the simulation code coming from the research team is not in good shape, including a detailed description, it cannot be used for this task and has to be re-coded to be suitable for the new design environment. Typically, the first step is thus re-creating existing code and verifying it with the old code. This requires multiple interactions between the research group and the implementation team. Once the implementation team is confident of having a good code reference, a so-called golden code, for all their later design stages they will start building an ASIC or programming a DSP. From here on, co-simulation for the purpose of verification is what mostly defines their work. It is therefore most helpful when a design environment supports co-simulation. Otherwise, two different tools need to be run in parallel ensuring identical input and output patterns. It is here where numerical problems of the algorithms show up. Typically, the hardware platforms onto which the algorithms are mapped are fix-point environments. Interaction with the research team

follows, defining which mathematical tricks can help to solve numerical issues. [6]

2.4. Automatic tool support

While the research team accomplishes its tasks with very few tools (a simple C compiler can be sufficient), there is hardly any tool support for the second team. Although some architectural exploration tools with very specific capabilities are available these days [7] there are no general tools that would support an automatic partitioning of a larger C code written by the research team into hardware/software (HW/SW) or arbitrary DSP/ASIC. The product realization team on the other hand, is faced with nearly hundreds of different tools for automatic design. Classically, such tools are divided into two categories depending on the hardware they support: DSP chips and FPGAs/ASICs.

Using *commercially available DSPs* gives the advantage of low-cost, high-speed devices whose development time would be otherwise quite long. They offer highest flexibility since their program can be changed later even when the product has already been sold. Usually, assembler language translators are used for programming DSPs. The work of a DSP engineer is characterized by writing thousands of lines of assembler code. Today many DSP providers offer C-compilers, allowing programming DSPs in a high-level language. This speeds up productivity tremendously. However, a few misconceptions usually come along with such products. Typically, the C-compilers are not very efficient in mapping C code to the DSPs hardware architecture. Thus, the time-limiting, high-complexity tasks still need to be written in assembler. Second, a C-program written for simulation is typically written in a different, hardware independent style so that even a good C-compiler with optimizer cannot map it efficiently to a specific DSP core. Once assembler code is written, instruction set simulators (ISS) support co-simulation and verification of the algorithms.

The conditions for FPGA and ASIC development are almost completely antagonistic to those of the DSP world: slower development, higher cost in small numbers, and difficult to change codes after the product has been shipped. Nevertheless, some applications do not allow using DSPs only, such as CDMA front ends, with their very high demand for high-rate bit-level operations. ASIC and FPGA designs are much more time consuming than DSP development and not surprisingly numerous EDA tools are offered for them. Such tools start at a high abstraction level allowing graphically grouping functional blocks in such a way that they exhibit the same behaviour as the original code of the research group. From here, the code has to be refined into fix-point code and further on converted into such structures that can be mapped directly into hardware blocks. While the design flow in many tools requires VHDL (or Verilog) coding by hand, others offer automatic translation from high-level languages. [8] No matter what philosophy the designers follow, many more refinement steps are required and with every step of refinement co-simulations for verification of the blocks are required. While CoCentric System Studio now also offers co-simulation of VHDL and C-code, [9] this has not been a standard feature in the past. Once the refinement steps in the original code move from one language to another, the verification process becomes very time consuming, mostly due to a lack of interface techniques. Also, once this language step has been made, there is no backward compatibility any more. Thus, modifications in the VHDL code are not reflected in the simulation code any more and finding a bug can become very costly.

A further problem arises during cycle accurate co-simulation of software and hardware in embedded systems. While software simulations are usually carried out on appropriate high-level models of the processor/DSP platform, hardware simulations run in a cycle accurate simulator. Separate simulations of the two aspects often neglect or simplify the connecting parts of the system such as busses or common resources. This can lead to bus congestions or

[6] Or, like in the mentioned HiperLAN/1 example above, a complete re-design is required costing much more than initially expected.

[7] VCC from CADENCE, N2C from COWARE and Visual Elite from INNOVEDA who recently joined forces with MENTOR GRAPHICS [2,28,46].

[8] A|RT-Builder [2] from ADELANTE TECHNOLOGIES and the compiler of SystemC [22,27] Version 2.0 supported for example by CoCentric System Studio from SYNOPSYS.

[9] A process that is very time consuming since a complicated interface has to be written by hand.

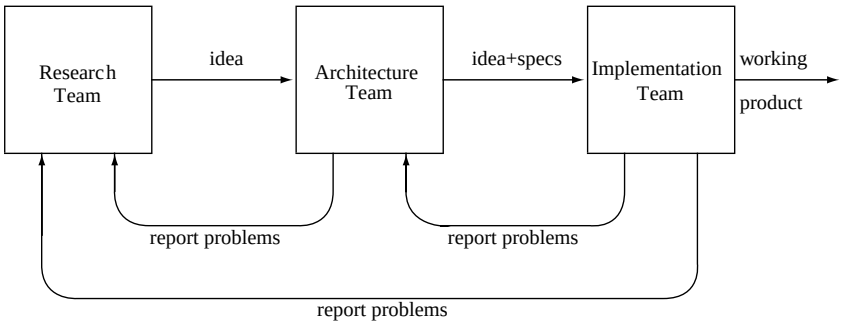

Fig. 1. The classical design flow.

violations of timing restrictions (such as maximum latencies).

3. Rapid prototyping: a new design flow

Note that the classical development approach, as it has been described in the previous section, often exhibits discontinuous phases. Typically, at the time when the design is handled from one team to the next, an entire new set of tools, languages and formalisms is used and the previous code cannot even be used for reference purposes. Moreover, once a problem occurs, a team cannot return their design to the previous team in order to get help in finding problems since the current code is not *backwards compatible*. The classic design flow is thus a feed-forward structure where the refined product and corresponding responsibility is only moved in one direction while necessary expertise and information revealing discussions are only made possible by backward loops (see Fig. 1).

Once unexpected problems occur, discussions with both (or even all three) teams are the typical method for salvation. Note that such a procedure is necessarily slow, due to the feedback loops allowing crucial information to come up only at the end of the workflow. What makes them unnecessarily slow is the fact that every time a problem occurs at a later design stage, this problem does not fit into the modelling of the preceding group. It can take a substantial amount of time in order to bring one of the teams up to the detailed knowledge level of the other one. In the following five points, areas of inefficient design are identified and immediate solutions are proposed.

(1) Clearly, the feedback loops cannot be broken since the required skill sets are not present in all teams. However, the reaction time can be changed dramatically, once all groups share one environment. This observation on its own is not new and many tools in the EDA community exist (COSSAP/CoCentric System Studio and SPW to name the two most widespread). However, since they have been developed to support specifically chip design (and to some extent algorithmic design) the architectural level and its exploration, as well as testing and system integration on specific hardware platforms (so-called platform-based design) are not supported. Also, due to specific language constraints many researchers refuse to use such systems, since they believe their productivity is dramatically reduced by them. On top of that, high prices with roughly $40k per license seat prevent many companies from using them throughout the whole design chain.[10] Clearly, the impact of selecting such a design environment is manifold and not only restricted to the prototyping, but has many implications for the entire process of product development.

(2) A second aspect is the missing documentation of the research team. While they focus on meaningful results of their simulations, the two other

[10] One may consider a design team with 200 members all working with such an environment. The total costs of $8,000,000 are supplemented by 15% annual maintenance costs. For a prototype design, however, the cost for a typically smaller group is not as significant.

teams are mostly interested in specifications. Graphical systems like COSSAP/CoCentric System Studio, SPW and SIMULINK offer the possibility to define functional blocks with clearly defined input and output ports and corresponding data rates. Using such a graphical system induces a documentation while specifying the functional blocks. Specification allows detection of flaws at an early stage and, much more importantly, it can be used to supply additional information into functional descriptions of algorithms. The graphical description has further advantages: it avoids global variables. Global variables can lead to the undesired effect that information from the transmitter and/or channel is known at the receiver and some (undesired) cheating can be the result. It is, for example, quite common in literature to present MMSE receivers having perfect knowledge of the signal-to-noise ratio (SNR), while this value in reality needs to be estimated. The problem of estimating such a value is typically underestimated. In addition to the graphical specification, COSSAP for example, allows writing so-called *Generic-C*, an ANSI C program enriched by essentially a header specifying the names, types and rates of the input and output variables, a feature well preserved in the so-called PRIM models of the new version CoCentric System Studio.

(3) A third aspect of the slowdown in the product flow is the required permanent re-coding. Although the research team defines a code for simulation, the system design team is not able to reuse the code, mainly due to its poor documentation and coding style used. The system realization team even rewrites the code to specify its own reference code (golden code). Based on the anticipated hardware platform, other languages (assembler, VHDL) have to be used at a certain level of refinement, requiring time-consuming hand re-coding. Such foregoing is error prone [11] and requires a solution based on automatic re-coding tools. While graphical tools do not provide an immediate solution to the tedious re-coding process, they can support it by allowing multiple, but different descriptions for each block. This alleviates the transformation and allows doing it piece by piece. SIMULINK [37], COSSAP/CoCentric System Studio [11] and SPW [38] allow for such code versions, but only on a block level, i.e., if a modification impacts several blocks at the same time, the system does not work out the required consistency. Revision control tools (such as CVS [20] or ClearCase [18]) can help here. At certain times in the design flow, the entire design becomes frozen and can be re-instantiated at a later time allowing to track a bug that shows up at a certain stage in the design flow, but was not noticed before. Further aspects of revision level tools are personal responsibility: the blocks can be assigned to specific people in the team and cannot be altered by others. Using such blocks, it is guaranteed that everybody in the team is working in the same, rather than in a personal environment. In order to guarantee backward compatibility it is important to stick with one code and the same language for as long as possible.

(4) Furthermore, graphical systems allow an easy method of code refinement by co-simulation. The code can be refined from one revision level to the next and by instantiating the two versions at the same time their output can be compared while they are fed by the same input. An important step in code refinement is the switch from float to fix-point code. The recent *SystemC* initiative [22,27] supports this step by extending ANSI C with fix-point data types. A|RT-Library [2] from ADELANTE TECHNOLOGIES offered such C++ Library extension for many years. The underlying idea is that by providing more and more details in a code at a very high level, the automatic tools modify the code iteratively into the required (meta-) descriptions until the final product is defined in every (technical) detail. These final descriptions express the code in a desired form, i.e., assembler code to program a DSP chip, VHDL or VERILOG code to program an FPGA or synthesize the required masks for an ASIC. However, since the automatic tools map the code from a high level to each of the lower levels, the refinement of

[11] See for example [24] where it is claimed that errors found late in the design process cost up to 100 times more than those found early.

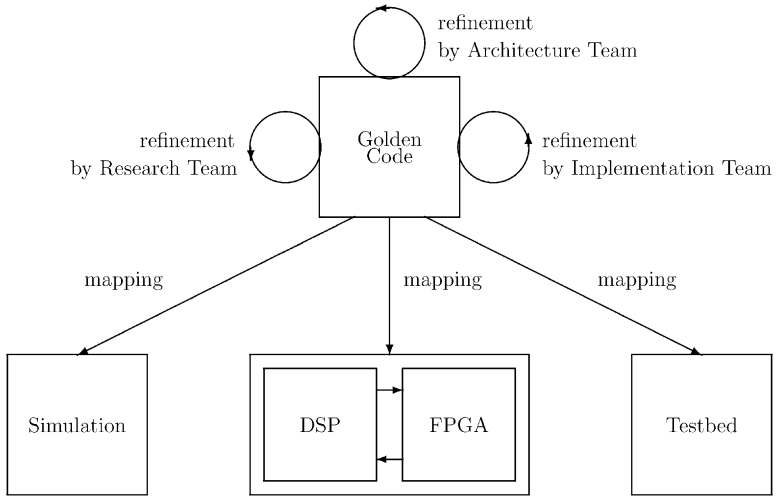

Fig. 2. The new design flow.

the code is only performed on the high-level description, i.e., the C-code. By iteratively rewriting the original C-code used in simulation to suit the needs of a specific hardware platform, the code remains backward compatible at any state and thus allows all teams to share the code and investigate problems. Specifically, there is no need to switch design environments when transferring from one team to another.

(5) One last aspect when analysing the slow development process is the team size. Poor communication is a drawback of rather large teams. Fortunately, the required amount of people in a prototype team is much smaller and it is possible to keep all team members as one team supporting full information to everybody. This is clearly a particularity in rapid prototyping that cannot easily be realized in a large product design team. However, improving communication can be a crucial point there as well and certainly requires consideration.

To summarize what has been learnt by investigating the current design flow, the key essentials to make prototyping rapid (and possibly speed up production), are given by our Five-Ones Approach:

- one environment
- one automatic documentation by specification
- one forward–backward compatible code revision tool
- one code to be worked on by refinement steps
- one team to improve communication

The proposed approach (depicted in the upper part of Fig. 2) at this stage, thus, allows a continuous flow in which the teams start with designs from the previous teams but stay within the same design environment. They gradually refine the existing routines in such a way that they still can be handed backwards for affirmation and for solving of specific problems.

4. Case study: MIMO-transmission over UMTS

So far a design flow paradigm (Five-Ones Approach) was proposed without specifying details about its actual implementation. Such details will follow in this section based on a case study. The goal of this section is thus to describe our experiences with the application of the proposed design flow in a real environment. Thereto the steps in the development and the prototyping of a multi-antenna wideband communication system are presented. The development process is explained step-by-step and it is demonstrated how the described unified design environment helped to achieve the projects goals.

4.1. Motivation

Recently it has been shown that systems with multiple antennas at the transmitter and at the receiver can utilize the spatial diversity in rich scattering environments extremely well to increase system capacity [12]. This is achieved by applying multiple-input–multiple-output (MIMO) techniques to perform spatial multiplexing. Thereby separate data streams are transmitted concurrently from each of the transmit antennas through a common resource, e.g. the same carrier frequency, spreading code, etc. This allows a direct increase in the system throughput. While this concept has been demonstrated for narrow-band systems [14,48] it has not yet been applied in practice to CDMA-based wideband communication systems such as the upcoming UMTS standard [44].

The goal of this project was to investigate how MIMO techniques could be incorporated into the UMTS downlink, to demonstrate that the chosen approach could be implemented efficiently, and to quantify the enhanced performance over the existing standard. In order to be able to make a clear assessment about the achieved performance improvements, it was important to adopt a system that inherits most of the essential parts of UMTS and that modifications were only applied where they would be absolutely necessary. A configuration with four antennas at both transmitter and receiver side was chosen for the realization.

In the following only the baseband part of the entire prototype is described. A radio front-end at 2.4 GHz was reused from previous BLAST experiments [1,14,48]. A detailed analysis of the expected behavior on algorithmic level can be found in [33,35]. Various channel estimation methods were also tested by applying a channel emulator (TAS FLEX4500) before the final measurements took place [1]. More details on the concept, the RF front-end as well as measurement results are reported in [1,9,16,33,35].

4.2. MIMO-transmission over UMTS

The system specification basically consisted of four almost independent UMTS-base station transmitters as depicted in Fig. 3. All of them contain the key components to support downlink communication:

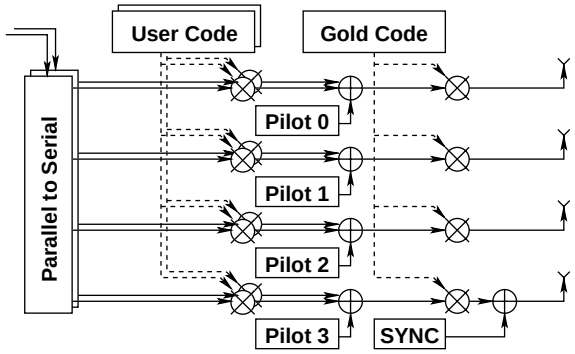

Fig. 3. Transmitter for a MIMO-extended UMTS-FDD Downlink with four antennas.

- Primary and Secondary Synchronization CHannels for initial synchronization of the receiver (PSCH,SSCH).
- A Common PIlot CHannel (CPICH) used for frequency offset and channel estimation.
- Dedicated Physical Data CHannels for multiple users (DPDCH) spread by the channelization code (Gold Code) and separated by orthogonal codes with variable spreading factor (User Codes).

In order to support MIMO channel estimation each of the transmitters has to be able to use a separate channelization code for its pilot channel. However, the data signal from each user is split into four parallel parts through serial to parallel conversion which are then transmitted from the four antennas using the same channelization code. The transmitters also need to be able to support dummy users to simulate real-life traffic loads on the network and to test the impact of multiple users on the system performance.

The receiver functions are essentially the inverse to the transmitter functions. However, much of the required information is not known at the receiver, and hence estimation of such unknowns has to be performed first. Details are depicted in Fig. 4. The implemented receiver functions are:

- Pulse shaping filter, matching the transmitter filter.
- Automatic gain control
- A synchronization unit for initial synchronization of frames and slots based on the PSCH and SSCH.

- A frequency offset and channel estimation unit based on the CPICH including a finger search algorithm.
- A detection unit for one user signal based on the DPDCH. Many different decoding algorithms (ML, VBLAST, MMSE, ZF) were to implement.

4.3. Previous experience

In a previous prototyping experience, a straight forward 1×1 UMTS downlink system was implemented [15]. For this project a parallel approach was taken in which the system modelling as a high-level MATLAB model and its prototype implementation in C were conducted in parallel. The goal was to keep the designs compatible and to allow an exchange of modules between the high-level simulation and the actual implementation. Strong differences in the implementation and partitioning of the high-level model and the actual implementation turned out to be the major obstacles. Different interfaces even on similar blocks prevented the exchange of functional units between the two designs. Implementation issues previously neglected in the high-level simulation setup turned out to be the main reason for discrepancies between the high-level simulation and the real-time implementation on the test-bed. As a result, the realization of a downlink equalizer revealed an extremely severe performance degradation for time varying channels than that expected from the simulations.

The conclusion from this work was that a parallel approach of a C-based prototype and a high-level model are extremely difficult and labour intense to keep synchronized over the entire development process. Following the suggested unified design methodology the development of a high-level model based mainly on MATLAB blocks was therefore abandoned. It was decided that all system components eventually going into hardware have to be solely written in the proposed *Generic-C* dialect, even if it was clear that the first drafts of the design would be used only for simulations.

4.4. A behavioural implementation

In our team it was decided to use SIMULINK as a graphical environment basically for financial reasons: a SIMULINK license costs roughly a tenth of a COSSAP/CoCentric System Studio or SPW license. However, another aspect was important as well: the acceptance of the team members. Since the team consisted mostly of research-oriented people, MATLAB was a preferred tool and SIMULINK as a derivative was seen as something as close to MATLAB as possible. However, the use of SIMULINK also has a few disadvantages. One major shortcoming in SIMULINK is the fact that incorporation of C-code using S-functions is rather complicated. Other than in COSSAP, only little help is given to program the rather complicated S-functions.[12] Another problem seems to be the abundant documentation that requires weeks of reading until one finds and understands the point of interest. It was therefore, decided to write a translation tool mapping Generic C-code into the SIMULINK environment. From here on productivity could be increased substantially and SIMULINK did not lack behind the much more expensive COSSAP tool. Using Generic-C, the program is separated into three phases: a header with declarations, an initialization phase, the actual program body and finally a post phase that runs after the program has ended. Since routines were targeted to run in endless loops in a later hardware solution, it was decided not to support the post phase, a phase that typically is only used to de-allocate memory or close previously opened files. Note that it is not a good style to open files or dynamically allocate memory in a program that is intended to run on a low-cost hardware platform. Such constructs should not be used in the simulation and if in- or output data are required to read from or write to a file, for verification purposes, for example, one can use standard library functions.

However, as the complexity of our design increased, a decrease in simulation performance was also observed. Since many parts were designed to run cycle-true on a per-cycle basis, i.e., roughly speaking one data element goes in and one comes out, by increasing the number of blocks, the simulation slows down. The scheduler algorithm is mostly occupied by itself and only small parts of the PC performance are devoted to the actual simulation. A common argument of MATLAB-opponents is that *MATLAB simulations are slow* compared to C implementations. This

[12] Complicated also because SIMULINK supports time discrete and continuous systems at the same time.

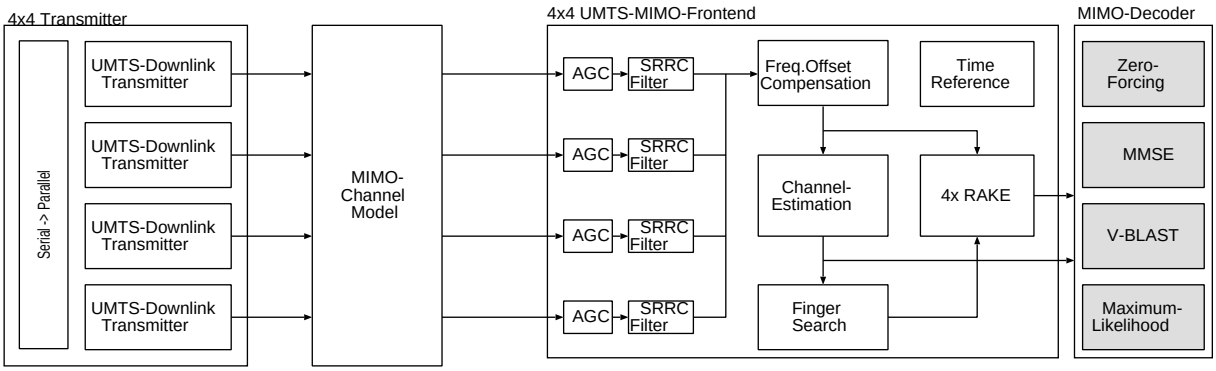

Fig. 4. Simulation set-up.

is mostly correct due to the nature of the MATLAB interpreter. Note however, that, SIMULINK allows programming all major parts in C. They are compiled before running and thus run as fast as conventional C-code. Nevertheless, SIMULINK has the reputation to run slower than conventional C-programs. This situation can be helped by two methods: using only few, larger blocks and presenting block operations at the input/output level of the blocks. If one, for example, increases the data size from one element to 1000 at the input of a block, the scheduler requires less than 0.1% of the complexity and can thus be neglected. In order to have the code prepared for such behaviour only a "for-loop" in C is required at the outer borders of the code block. With the Generic-C description it is possible to define the block sizes as a parameter that can dynamically be set at run time and be controlled by a MATLAB script, while at execution time on a hardware test bed the parameter can be set to one. The internal code remains unchanged. It is important to keep in mind that block operations in this context are typically for simulation speedup only and have no representation in the actual implementation where only single data items are available at the input of a design entity.

4.5. A coarse partitioning

The first SIMULINK design set-up included some purely behavioural blocks in floating point such as the channel model, a synchronization model and a module to perform BER-measurements. On the other hand it contained the actual MIMO transmitter and receiver. The former was composed of four of the transmitters originally designed for the 1×1 system. The receiver was partitioned into two major blocks, a receiver front-end and a MIMO-decoder back-end connected by a small module to emulate the interface between them. This partitioning was already chosen with respect to a future mapping of the components onto hardware. An analysis had shown that the front-end had to deal with high-rate data path type operations and would therefore be likely to be mapped to an FPGA. The MIMO decoder on the other hand would have to deal with highly data dependent and fairly irregular operations at a much lower rate and would therefore be much better suited for a DSP implementation. A detailed overview of the major functional blocks in the simulation set-up is depicted in Fig. 4. Note that the entire design is feed-forward oriented. By avoiding feedback loops, the number of interfaces are reduced also easing the partitioning of a system.

The definition of a graphical simulation set-up already contained an essential part of the basic interface definitions between the functional blocks of the system. With this as a starting point each block was assigned to a member of the development team. The remaining part of this case study will focus on the implementation of the MIMO receiver, consisting of the front-end and the decoder module.

The next step towards the realization of the system was to fill the still empty blocks with an initially purely behavioural description of their basic functionality. This was essential to enable the team to start system simulations and to make first performance

assessments. Only very little attention was paid to the suitability of the written code for future hardware implementation at this point. As multiple people started contributing to the development of single blocks (especially the front-end) functionality needed to be split further into hierarchical blocks. This was done through the definition of function calls within the Generic-C code. With this step-by-step partitioning it was possible to leave the top-level set-up mostly unaltered. It also helped to avoid the previously discussed problem where a growing number of blocks puts more load on the systems scheduler and therefore slows down simulations. Gradually *all* top-level Generic-C code was converted into well-defined functional blocks.

4.6. Impact of code revision

A revision tool (CVS) was introduced at this stage. It turned out later to be a very useful step. One person had to take on the responsibility of module testing and revision control. His tasks were to check whether new versions of functional blocks were compatible to previous ones. To achieve this goal, a reference environment with test vectors was created that could be repeated over and over once new revisions came in. Often it turned out that some designer also made changes in functional blocks other than his own block. In order to accept a new revision of a functional block in the CVS system, it had to undergo such testing by an independent developer. Once the tests succeeded, the new revision was introduced to CVS and all designers were informed to download the newest version. Due to the previous testing, this procedure went rather smooth. Simulations carried out with this set-up, showed promising results and confirmed that an implementation of the design in hardware would indeed be successful.

4.7. Hardware set-up and implementation

As a rapid prototyping platform the approach from SUNDANCE [41] was used allowing for a Lego-like system with different modules carrying FPGAs, DSPs, ADCs and DACs. Slow 8bit communication ports (up to 5 Mbps) and faster SDB ports (up to 200 Mbps) are supported on most modules. By writing a limited set of drivers for all possible combinations, one can impose the required driver at the time the system is being assembled. The 3L-Diamond RTOS[13] for the DSPs also allows the user to map the procedures onto several chips, i.e., the prototyping engineer can decide at a later stage how many DSPs are really required to meet real-time requirements.

The chosen hardware platform from SUNDANCE consists of a flexible combination of TI 'C6x series digital signal processors (DSPs) [43] and XILINX Virtex series FPGAs [45]. It is based on carrier boards with a PCI bus interface and a variety of high performance DSP and FPGA modules. A typical PCI carrier board supports up to four modules. In our final set-up one FPGA module, one C-67 DSP module and evaluation modules from other vendors for data conversion (ADC and DAC) were used. Originally, a much larger amount of FPGA modules was expected to be necessary. Since SUNDANCE provides a rather flexible and scalable platform, the final partitioning into hardware elements could be delayed to a very late point in time.

4.8. Further partitioning

A tremendous advantage of a graphical environment is that it already proposes a *partitioning* for various functional units that can aid the design specification group to make their decisions. It also provides first interface definitions with data rates between the already defined blocks. Partitioning was performed on two levels. A first partitioning identified large functional blocks like transmitter, and receiver front-end and decoding algorithms since they were predestined for specific hardware solutions: transmitter and receiver front-end for FPGAs and decoding algorithms for DSPs. At a much later stage a refined partitioning was required since it was initially not clear how many DSPs and how many FPGAs were required. Since the design realizes a feed-forward data-flow avoiding feedback loops as much as possible, cutting larger blocks into smaller functional units was straightforward. On the other hand if the design specification team decides to incorporate two or more blocks into one, it is simple and not very time-intensive to follow up with this new partitioning scheme in SIMULINK.

[13] Diamond is a trademark of 3L LTD.

Since no automatic partitioning tool was available, the partitioning was done purely based on experience. The receiver front-end was targeted for a XILINX Virtex-1000 FPGA, while the MIMO-decoder went to a TI-C62 or C67 DSP. An initial purely automatic mapping of the original code allowed first assessments about the area and speed of the FPGA design and the speed of the DSP code for the MIMO-decoder (through profiling). As expected the original implementations fell far short of their real-time (e.g. DSP cycles) and hardware (e.g. FPGA "slices") requirements. However, it helped to identify the major bottlenecks. These blocks were then optimized individually, replacing gradually their original behavioural implementations. The impact of any modifications was always limited to the smallest possible set of blocks: this ensured that the remainder of the design was not affected and that the system's functionality could be verified against the original set-up at any time. Changes on the top-level simulation set-up and on the interfaces were avoided as much as possible. Whenever a configuration was available that was fully functional and showed comparable performance to the previous design step it was checked into the CVS revision control system and a label was attached to the corresponding version of all the involved files. Using this gradual refinement strategy basically *all* modules of the receiver front-end were revised to achieve a design that could be implemented and would meet the area and timing constraints. While some blocks such as the timing reference, the filters and the gain control were only slightly modified during this process other blocks were revised completely or even rewritten.

Probably the best example for the importance of this refinement step is the channel estimation block. Its purpose is to estimate the channel profiles between each transmit and receive antenna. For the 4×4 system presented here this results in 16 channel estimates running in parallel. The original implementation of this block required more than 300% of the area available in the targeted Virtex-1000 FPGA. The optimized final realization was able to perform the same operation with an area consumption of less than 20% of the device. This was achieved through an architecture optimization of the channel estimator itself combined with a careful selection of the utilized channelization codes of the pilot channel (see also [9] for further details).

At this point the receiver front-end design had been fully partitioned into sub-functions which completely encapsulated all its functional units and provided well specified interfaces between them. Modifications within these blocks were entirely independent of the rest of the design. All blocks could be synthesized separately allowing to accurately estimate their complexity individually and to decide where further rework was needed.

4.9. Refinement and optimization

The next logical step was to start refining and optimizing the code for its implementation on the platform. Part of the specification phase is to define the *numerical conditions* under which the algorithms have to run. For example, if a 16 bit DSP has been selected to run a specific algorithm, it must be checked that this algorithm is numerically stable under such conditions. SIMULINK like other tools allows including a fixed-point block-set, i.e. graphical building blocks with simple operations like add, mult, etc. in which the length of each mantissa can be defined. This allows testing numerical sensitivity of an algorithm. However, reprogramming an algorithm from C code into its graphical form is time consuming [14] and the actual fixed-point simulation run becomes very slow. Run times increasing by a factor of 1000 were experienced! Therefore, it was decided not to use such fix-point blocks. Instead the evaluation was done through further refinement steps of the original floating point code. Two different paths were followed depending on whether the targeted hardware platform was a DSP or an FPGA.

4.9.1. Refinements for DSPs

As long as the specified hardware platform is a 16 bit DSP, the numerical analysis can be done in C by using short data types throughout the program. Possible shift operations have to be added in order to avoid over- or underflow operations. The advantage is that—assuming a good C-compiler exists—this program can be used directly to run on the target DSP. It

[14] For dataflow dominated code, a recoding may go fast but as soon as time-critical control flow dominated code is present, re-coding is very labour intense.

Table 1
Computational complexity of VBLAST and ML

		ZF-VBLAST			ML		
	Rate	DIV	MUL	ADD/CMP	DIV	MUL	ADD/Cmp
Per channel estimate	$\frac{3.84e6}{1024}\frac{1}{s}$	4	264	640	—	96	232
Per symbol	$\frac{3.84e6}{32}\frac{1}{s}$	—	160	104	—	64	635
Total	$\frac{\text{Ops}}{s}$	1.5M	20M	15M	—	12M	78M

was therefore decided to use TI's C62/67 DSPs since they are essentially 16 bit machines reflecting most DSPs on the market but at the same time an excellent C-optimizer exists due to its RISC structure (see for example [26]). It allows to reuse the already written C-code as opposed to switching to an assembler description. This is perfectly along the lines of the paradigm of using only *one code* since the C code can be reused without any changes. Also the use of an ISS was avoided entirely.

Although functionally correct the now available fix-point code may not be suited to be implemented directly on a DSP or ASIC/FPGA. DSPs, like the TI C62/67 support C-code but in various quality depending on how it is written. Several style elements can speed up the program in a DSP considerably. The read and write operations, for example, often require several cycles to perform and run better when pipelined. One method to improve this is mapping a *short* (16 bit) array into an array of type *int* (32 bit) and applying the algorithm to the lower and higher part separately. Another approach is applying INTRINSICS, i.e. specifically defined operations that will be mapped directly into one assembler operation. Once such INTRINSICS are used, the original code would not run in the SIMULINK environment since they are not defined in ANSI C. However, an include file with the corresponding definitions in ANSI C can be added under SIMULINK allowing bit-exact co-simulation. Therefore, the Generic-C mapping algorithm was extended to also support DSP platforms as a target. Table 1 depicts profiling results for two decoder algorithms (ZF-VBLAST and ML). By refining and optimizing the code it was possible to run a code rate of 32 for both decoders. This was a quite surprising result since the ML decoder, in particular, was expected to be of much higher complexity. Further refinement by using assembler language was not expected to improve the results significantly.

4.9.2. *Refinements for FPGAs*

The standard C-types such as char, short, and int are not a good way of specifying variables when the algorithm target is an FPGA (or ASIC). One would have to use the next larger value (for a 9 bit variable, a *short* data type is required) combined with a masking of the left over bits. However, this requires major code modifications and adds significant overhead for the designer. A much better approach is the one from ADELANTE TECHNOLOGIES [2], now also present in Open-SystemC. They provide a fixed-point library set that enriches the ANSI C-code by fix-point types for arbitrary mantissa length. Together with the library comes a statistic tool set that allows checking how often the bits in the mantissas are used so that the designer can quickly decide whether the mantissa is too short or too long. The advantage here is that the functional part of the original C-code can remain unchanged, while only the type declarations need to be re-specified. Once the final mantissa length is found, the code runs only about 10 times slower than the original code, still fast enough to make a decision on the numerical feasibility of an algorithm. It is also very important that the data types allow for arbitrary placement of the binary point so that one can switch directly from a floating to a fixed point description without additional shift operations. This process was automated by defining intermediate data types that can be mapped to float first and later to individual fix-point types. Using this procedure, the existing code was refined step-by-step from float to fix without changing anything but the data types. Note that this approach is still time consuming and one wishes to apply an automatic, iterative scheme like the one

proposed in [30] where the mantissa length can be found automatically.

4.10. Final steps towards implementation

Although from the start of the design the input and output port definitions were specified as part of each graphical block, these definitions were not sufficient when mapping the modules onto hardware. Note that all input and output definitions are served by block-oriented communication drivers. Depending on which hardware is expected to perform the communication link, different drivers are available and are specified when calling the mapping algorithm. As a standard for DSPs (by using the port name COM#), the communication ports (8 bit parallel ports) of the 3L-Diamond operating system are used which support the user with a complete (but hidden) DMA and interrupt structure for handling the data communication. Optionally, drivers for serial ports were written to offer a voice connection to the outside world. The automatic mapping tools were set to include a driver routine according to the name of the port once the keyword COM does not occur in the port name specification. The mapping of the DSP code into the 3L-Diamond supported RTOS code is also done automatically by the already mentioned mapping tool and indicated in Fig. 2 by the mapping in the centre of the figure.

Furthermore, the mapping tool also allows mapping an algorithm from Generic-C to DSP for test purposes. By imposing real-sync [25], it is possible to feed the DSP with the data that come out of the SIMULINK simulation, and once the results are present at the output of the DSP are fed back to the SIMULINK environment to continue the simulation process. By this, DSP testing is possible with the exact data as used in the simulation and without explicitly defining test vectors. Results can be compared with simulation while running and mismatches can be indicated and analysed. Such a set-up can also be used to speed up simulations. The corresponding mapping is depicted in Fig. 2 on the right branch.

Mapping algorithms into the FPGA world requires additional steps. As explained before, the ANSI-C code is enriched by fix-point data types from ADELANTE TECHNOLOGIES. Once the code runs entirely in fixed-point C data types, the algorithm can automatically be converted into VHDL by applying the A|RT-BUILDER tool from ADELANTE TECHNOLOGIES. From there the VHDL synthesizer continues to map the code into the required format for FPGAs. Note that this design path requires the developer to write the C code in a specific style. The code must represent a Finite State Machine (FSM) so that the code can be mapped into a unique input–output mapping. Time-consuming operations must be pipelined by hand. All control logic needs to be added by the designer. [15] Connecting the hardware ports is supported by SUNDANCE by providing VHDL code specifically for them. Other than for the DSPs, they need to be imported at a later stage when the design is present in VHDL. Some FPGAs support specific core elements like memory, multiplier or pre-synthesized and optimal structures. These can also be included by defining them as separated C-code routines and instantiating them with the following statement:

```
#ifdef __SYNTHESIS__
dummy c-code procedure (to be replaced
automatically later on)
#else
behavioral simulation code
#endif
```

During the simulation the behavioural model is used. When mapping to VHDL the dummy component is replaced by the actual core element. The optimized FPGA design was automatically mapped to VHDL synthesized and placed. As memory macros were used in the design, behavioural descriptions had to be replaced by the corresponding VHDL templates. This was done automatically applying script files before synthesis. While this translation step worked quite smoothly another 3 weeks of work was necessary to work around interfacing and clocking issues on the platform that were not included in the system simulations. However, after these problems were solved, the basic functionality of the design turned out to be working without requiring major modifications in the *Generic-C* code.

[15] Modern EDA tools like A|RT-DESIGNER (see [2,32]) allow even to add the control logic to a given C code description.

4.11. Measurement-based system improvements

After the successful implementation of the system, a first set of measurements could be performed. A debug interface in the receiver front-end allowed monitoring critical internal parameters of the system. It was designed to help locating and identifying problems and to make measurements of parameters that would otherwise be hidden inside the receiver. This system for example helped to identify a problem in the frequency offset estimator. The failure was traced back to an inaccuracy caused by a linear approximation of the arctan(x) function. As this approximation error would only show up for certain frequency offsets and only at certain situations, it had not been found during simulations. The observations from the real-time experiments were fed back to the designers and an improved slightly more complex approximation removed the problem [9].

5. Conclusions and outlook

Noting the different constraints every design team faces, many rapid prototyping approaches have been proposed and successfully used. The following publications describe approaches which have similarities to our approach. In [42] a system level design on algorithmic level using SIMULINK with its RTW feature is proposed to speed up simulation by programming in C. In [36] an approach very similar to ours using SPW rather than SIMULINK has been proposed. However, the features of SPW and the lack of an automatic C-to-VHDL conversion tool along with the utilization of TI C40 processors does not allow a consistent language description throughout the design steps. A more focussed view on prototyping for wireless systems based on pure FPGA programming is presented in [8]. Utilizing SIMULINK graphical block descriptions corresponding parametric VHDL code is provided to map a SIMULINK design directly into a set of FPGAs. How the intercommunication of such FPGAs is supported and how the partitioning of a rather large design spanning over many FPGAs is performed is not mentioned. Finally [39] should be mentioned as one example of the future challenges in designing reconfigurable hardware for so-called software-defined radios.

By observing the similarities between building prototypes and building products, the overall design flow was improved by combining the successful strategies previously employed separately. A review of the various steps in a product design flow identified several aspects responsible for slowing down the design process. These pitfalls are prevented in a prototyping environment by the proposed Five-Ones Approach: one team, one environment, one code, one documentation and one code revision tool. A detailed description for automating these steps is presented leading to a further increase in productivity. Many of these techniques are directly applicable to product design flows. The reported case study for MIMO transmission over UMTS required about 10 man-years of effort for handling a complex design that could otherwise require 5–10 times more effort.

Some points still need to be addressed in the future to speed up such processes. Embedded systems are emerging into the wireless market and need to be addressed in the rapid prototyping environment as well as in the product design environment. Many companies build their own prototyping platforms with specifics to suit their product line. While our realized rapid prototyping flow is based on supporting SUNDANCE boards, a more general approach must allow a parametric specification of such platforms by including their interface information. The work environment with its mapping tools, though quite capable of handling larger designs, is currently being enhanced in order to allow for new features. One of these is automatic fix-point coding. As described in [30] the tools should be able to find out their own minimum mantissa lengths. Architectural exploration by incorporating A|RT-Designer from ADELANTE TECHNOLOGIES [2] is another exciting feature. Finally, automatic partitioning as proposed in [6] and supported in newer EDA tools [28] would be a salient feature in such an environment.

Acknowledgements

The authors would like to thank the entire prototyping team, in particular, Sue Walker, Michele Milbrodt, David Haessig, Dragan Samardzija, Maxime Guillaud, Salim Manji, Goran Djuknic, Suman Das, Veronique Battier, Thomas Gvoth, Ali Adjoudani, as

well as Paul Polakos, Tod Sizer, Stephen Wilkus, and Arnold Siegel for their support during tough financial times. We would also like to thank Brian Jones from Rice University who provided us with his real-sync software and finally Pavle Belanovic for carefully proof-reading the paper.

References

[1] A. Adjoudani, E. Beck, A. Burg, G.M. Djuknic, T. Gvoth, D. Haessig, S. Manji, M. Milbrodt, M. Rupp, D. Samardzija, A. Siegel, T. Sizer II, C. Tran, S. Walker, S.A. Wilkus, P. Wolniansky, Prototype experience for MIMO BLAST over third generation wireless system, Special Issue JSAC on MIMO Systems, 21 (April 2003) 440–451.

[2] A|RT Builder, A|RT Designer, and A|RT Library are trademarks of ADELANTE TECHNOLOGIES, http://www.adelantetech.com.

[3] R. Baines, The DSP bottleneck, IEEE Comm. Mag. 33 (5) (1995) 46–54.

[4] R. Baines, D. Pulley, A total cost approach to evaluating different reconfigurable architectures for baseband processing in wireless receivers, IEEE Comm. Mag. 41 (1) (January 2003) 105–113.

[5] Y. Baltaci, I. Kaya, A. Nix, Implementation of a HiperLAN/1 compatible CMF-DFE equaliser, Proceedings of VTC, 2000, p. 1884.

[6] T. Blickle, J. Teich, L. Thiele, System-level synthesis using evolutionary algorithms, Design Automat. Embedded Systems (1) (1998) 1–40.

[7] U. Bortfeld, C. Mielenz, White paper C++ system simulation interfaces, Infineon, July 16, 2000.

[8] R.W. Brodersen, W.R. Davis, D. Yee, N. Zhang, Wireless systems-on-a-chip design, in: Proceedings of the International Symposium on VLSI Technology, Systems, and Applications, Hsinchu, Taiwan, April 2001, pp. 45–48.

[9] A. Burg, M. Guillaud, M. Rupp, E. Beck, D. Perels, N. Felber, W. Fichtner, FPGA implementation of a MIMO receiver front-end for UMTS, Proceedings of the Zürich Seminar, February 2002, pp. 8_1–8_6.

[10] A. Burg, B. Haller, M. Guillaud, M. Rupp, E. Beck, L. Mailaender, A rapid prototyping methodology for algorithm development in wireless communications, in: Proceedings of Design, Automation and Test in Europe DATE'01, Munich, 13–16 March, 2001.

[11] Design Compiler and COSSAP/CoCentric System Studio are trademarks of SYNOPSYS INC., http://www.synopsys.com.

[12] G.J. Foschini, M.J. Gans, On limits of wireless communications in a fading environment when using multiple antenna, Wireless Personal Comm. (6) (1998) 315–335.

[13] J. Gleick, A bug and a crash, http://www.around.com/ariane.html.

[14] G.D. Golden, G.J. Foschini, R.A. Valenzuela, P.W. Wolniansky, Detection algorithm and initial laboratory results using V-BLAST space-time communication architecture, Electron. Lett. 35 (1) (January 1999) 11–14.

[15] M. Guillaud, A. Burg, L. Mailaender, B. Haller, M. Rupp, E. Beck, From basic concept to real-time implementation: prototyping WCDMA downlink receiver algorithms—a case study, Proceedings of 34rd Asilomar Conference, Monterey, CA, October 2000, pp. 84–88.

[16] M. Guillaud, S. Das, A. Burg, M. Rupp, E. Beck, Rapid prototyping design of a 4×4 BLAST-over-UMTS receiver, Proceedings of the 35th Asilomar Conference, Monterey, CA, November 2001, pp. 1256–1260.

[17] A. Hoffmann, T. Kogel, H. Meyr, A framework for fast hardware–software co-simulation, in: Proceedings of Design, Automation and Test in Europe DATE'01, Munich, 13–16 March, 2001, pp. 760–764.

[18] http://www.rational.com/products/clearcase/index.jsp.

[19] http://www.upv.es/cost259/right.htm.

[20] http://download.cyclic.com/pub/.

[21] http://www.proxim.com.

[22] http://www.systemc.org.

[23] W.C. Jakes, Microwave Mobile Communication, IEEE Press, New York, 1974.

[24] R.S. Janka, Specification and Design Methodology, Kluwer Academic Publishers, Dordrecht, 2002.

[25] B. Jones, S. Rajagopal, J. Cavallaro, Real-time DSP multiprocessor implementation for future wireless base-station receivers, TI DSPS Fest, Wireless Applications, TX, August 3, 2000.

[26] A. Klauser, Trends in high-performance microprocessor design, Telematik (1) (2001) 12–21.

[27] W. Mueller, J. Ruf, D. Hoffmann, J. Gerlach, T. Kropf, W. Rosenstiehl, The simulation semantics of systemC, in: Proceedings of Design Automation and Test in Europe, DATE'2001, Munich, 13–16 March, 2001, pp. 64–70.

[28] N2C is a trademark of CoWare, http://www.coware.com.

[29] J. Nikolic-Popovic, Application of Texas Instruments' TMS320C6400 in 3G wireless infrastructure transceivers, Proceedings of the Ninth Signal Processing Workshop, Hunt, TX, October 2000.

[30] O. Ogawa, K. Takagi, Y. Itoh, S. Kimura, K. Watanabe, Hardware synthesis from C programs with estimation of bit length of variables, IEICE Trans. Fund. E82-A (11) (November 1999) 2338–2347.

[31] V.N. Patel, C.K. Wiese, F.M. Hiemstra, S.C. Himes, Rapid development and commercialization of products-A business imperative in the global telecommunication landscape, Bell Labs Technical J. 5 (4) (October–December 2000) 157–171.

[32] M. Rupp, A 64-point FFT design example using A|RT-designer, Proceedings of the 34rd Asilomar Conference, Monterey, CA, October 2000, pp. 389–393.

[33] M. Rupp, On the impact of uncertainties in iterative MIMO decoding, Proceedings of the 36th Asilomar Conference, Monterey, CA, November 2002.

[34] M. Rupp, E. Beck, R. Krishnamoorthy, Rapid prototyping for a high data rate wireless local loop, Proceedings of the 33rd Asilomar Conference, Monterey, CA, October 1999, pp. 993–997.

[35] M. Rupp, M. Guillaud, S. Das, On MIMO decoding algorithms for UMTS, Proceedings of the 35th Asilomar Conference, Monterey, CA, November 2001, pp. 975–979.
[36] I. Seskar, N.B. Mandayam, A software radio architecture for linear multiuser detection, IEEE J. Selected Areas Comm. 17 (5) (May 1999) 814–823.
[37] SIMULINK is a trademark of MATHWORKS, http://www.mathworks.com-1888, 2000.
[38] SPW is a trademark of CADENCE CORPORATION, http://www.cadence.com.
[39] S. Srikanteswara, J.H. Reed, P. Athanas, R. Boyle, A soft radio architecture for reconfigurable platforms, IEEE Comm. Mag. 38 (2) (February 2000) 140–147.
[40] Y. Sun, A.R. Nix, D.R. Bull, D. Milford, H. de Beauchesne, R. Sperling, Ph. Rouzet, Design of a novel delayed feedback equaliser for HiperLAN/1 FPGA implementation, VTC 1999, pp. 300–304.
[41] Sundance is a trademark of SUNDANCE MULTIPROCESSOR TECHNOLOGY LTD. & SUNDANCE DSP INC., http://www.sundance.com.
[42] V. Sundaramurthy, J.R. Cavallaro, A software simulation testbed for third generation CDMA wireless systems, in: Proceedings of the Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, October 1999, pp. 1680–1684.
[43] TI TMS320C6000 DSP Platform http://www.ti.com/sc/docs/products/dsp/c6000/index.htm.
[44] UMTS-Standard: TS 125.211, TS 125.213, ETSI, http://www.3gpp.org.
[45] Virtex and ChipScope are trademarks of XILINX INC, http://www.xilinx.com.
[46] VISUAL ELITE is a trademark of INNOVENDA, http://www.innoveda.com/products/datasheets_HTML/visualelite.asp.
[47] J.S. Wang, P.L. Lin, W.H. Sheen, D. Sheng, Y.M. Huang, A compact adaptive equalizer IC for Hiperlan system, ISCAS 2000, Switzerland, May 2000, pp. II265–268.
[48] P.W. Wolniansky, G.J. Foschini, G.D. Golden, R.A. Valenzuela, V-BLAST: an architecture for achieving very high data rates over rich-scattering wireless channels, in: Proceedings of the ISSSE-98, Pisa, Italy, 1998, pp. 295–300.