EXPERIENCES WITH A VERSATILE PROTOTYPING PLATFORM FOR TOP–DOWN HARDWARE AND SOFTWARE DESIGN FOR COMMUNICATION PROTOCOLS

Andreas Iselt and Andreas Kirstädter

Technical University Munich, Institute for Communication Networks
Arcisstr. 21, D–80290 München, Germany
e–mail: iselt@lkn.e–technik.tu–muenchen.de

ABSTRACT

This paper discusses a new prototyping platform for effective protocol design and rapid prototype implementation. From specification to implementation all steps are supported by semi–automatic tools that just require the design specification itself plus some directives and constraints. For the hardware implementation of a prototype, a programmable core consisting of FPGAs is realized. Thus quick changes at the redesign of a system are possible. The hardware and the design software of the prototyping environment are modular and allow an easy substitution or extension of the parts of the system. A demonstration application (of a FDDI–on–SDH remote repeater) shows the functionality and effectiveness of the platform. The paper concludes with an extensive discussion of the experiences gained with the chosen top–down design approach for high–speed reprogrammable logic.

I. INTRODUCTION

The increasing transmission capabilities of fiber–based networks provide the possibility to deliver broadband services (e.g. at the STM–1 level of SDH, 150 Mbit/sec) with corresponding low latencies to the customer. Resulting from the processing speed requirements the lower layers in the data path have to be implemented more and more as (pure) hardware solutions.

Another development is the globalization of markets together with the resulting increased level of competition. They make a fast time–to–market and the ability to quickly react to protocol changes prerequisites for the survival of manufacturers. Often completely new services have to be implemented (and sold) before their standardization has been completed. Examples are the Available Bitrate Service (ABR) and the policing function (User Parameter Control, UPC) in ATM networks. At the extreme case, modifications of the high–speed protocols (and not only bug fixes) have to be implemented into equipment already in use at the customer's site.

Thus on the one hand, reprogrammable hardware components (e.g. Field Programmable Gate Arrays, FPGAs) are used more and more frequently in the designs. They not only provide a shorter turn–around time than ASICs (together with a lower level of fixed costs) but also the ability to adapt the designs to protocol modifications by simply updating their configuration memories. This update could even be done via the communication network itself.

While the usage of FPGAs is nearly state–of–the–art for the control part within those high–speed protocol implementations, the corresponding data path is still commonly implemented using conventional fixed logic: the required speed for operations like barrel–shifting, cell/packet filtering/insertion, and buffering is considered to exceed the capabilities of a pure FPGA–based solution. Undoubtedly the resulting hybrid (reconfigurable + fixed logic) architectures are quite unsatisfying because of their inflexibility.

On the other hand, short turn–around times also require the strict usage of top–down design techniques and hardware description languages (with their advantages of modularity and reuse of components). Furthermore, the integration of synthesis techniques for reprogrammable logic components into common CAE environments has reached a state where a realistic top–down design of reconfigurable logic is possible.

Therefore at our institute for communication networks a flexible and modular top–down prototyping system for high–speed protocols has been designed and tested. This prototyping system not only serves the purposes of accelerating and simplifying the processes of design, evaluation, and implementation of future high–speed communication protocols. It has also been designed (together with the demonstration application of a remote FDDI–on–SDH repeater) to gain experiences concerning the design of complete (i.e. including the full data path) communication protocols near the speed limits of the underlying FPGAs.

In contrast to other known approaches (see. e.g. [10], [11], [12] for an discussion of the VLSI "suitability" of communication protocols and [4], [5], [6], [7], [8], [9] for the usage of parallel processing platforms) our platform focuses more on the investigation of speed–critical parts and the verification of new protocol mechanisms. Thus the platform itself is only applicable to small protocols and protocol conversion systems (such as e.g. the application described in section 4). But the experiences and results (concern-

ing both protocol mechanisms and the top–down approach itself) can easily be applied to large systems as well.

After design entry (via the hardware description language VHDL or a schematic) and subsequent functional verification the protocol (including the full data path) is automatically synthesized for the FPGA architecture at the target prototyping platform. This platform consists of several modules:

- a high–speed core (XILINX FPGAs),
- electro–optical interfaces (at speeds of 125 and 155,52 Mbit/s), data parallelization/serialization, SDH framing, and
- a microcontroller for configuration (of the FPGAs and the SDH framer) and low–speed management tasks.

For the design of the firmware and application software a remote debugging option (on source–code level) was established.

The next section describes the modules of the hardware platform together with the cross compilation system and the remote debugging environment. In Section 3 the top–down design approach and the applied tools are shown. Section 4 explains the remote FDDI–on–SDH repeater implemented as a demonstration application. Finally results and experiences are discussed in Section 5.

II. HARDWARE PLATFORM

Figure 1 gives an overview of the hardware part of the protoyping system. It consists of a programmable hardware core for the high speed data path and control operations, and an optical 125 Mbit/sec interface as it is used for FDDI or ATM–TAXI. Further, a SDH framer for STM–1 (155.52 Mbit/sec) is included. The SDH interface can be externally connected to optical and electrical lines. For management purposes and low speed tasks of the application a microcontroller is provided.

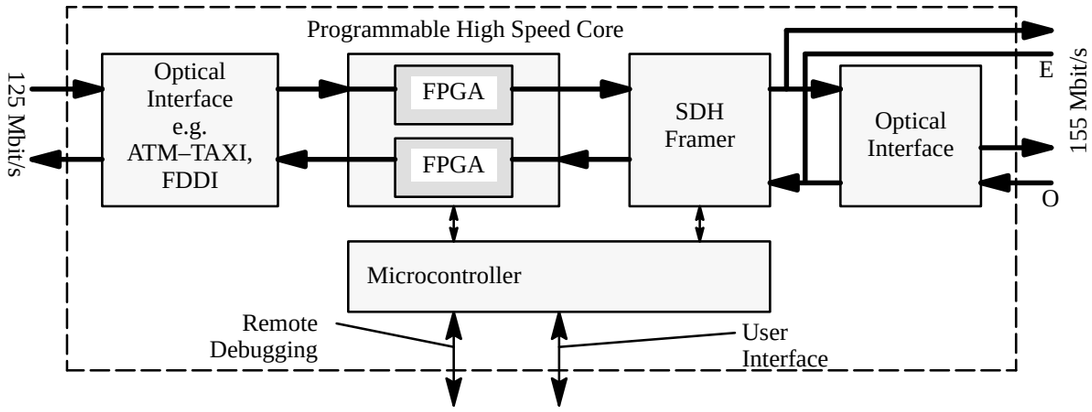


Figure 1 : *Hardware platform of the prototyping environment*

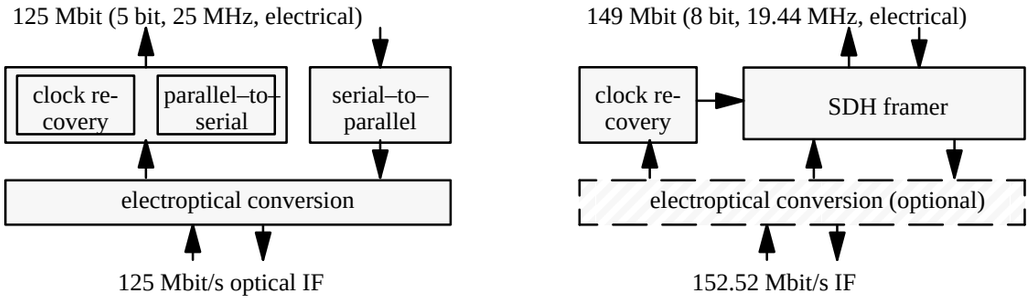
The hardware concept is very modular. A passive backplane provides at least 4 identical sockets for the insertion of modules. At the moment, there are 3 modules available. One module consists of the microcontroller with its peripherals. Another module combines the high speed interfaces. The programmable high–speed core is also located on a board of its own. The modular implementation enables an easy incorporation of new modules and substitution of existing modules. For example, the interfaces are located on a plug–in card, that can be easily removed from the interconnection backplane, or replaced by a different board, incorporating new interfaces.

Programmable High Speed Core

The programmable high–speed core is currently built of two XILINX XC4010E–3 FPGAs. Specified at gate delays of 3 nsec they support system clock frequencies up to around 25 MHz (because of rather high additional internal routing delays). Each XILINX FPGA can realize roughly 10,000 gates. For most protocols and protocol conversions this should be sufficient. Especially having in mind, that slower control tasks can easily be outsourced to the microcontroller, and only the high performance data path operations have to be done in the FPGAs.

FPGA programming has been implemented using the microcontroller and its standard EEPROM based memory. At power up or on user request, the microcontroller loads a new configuration actively from the memory to the FPGAs. As described below the transfer of configuration data from the development workstation to the microcontroller’s EEPROM via a

serial link is supported very comfortably by the microcontroller firmware.



a) *Optical Interface (FDDI, pure ATM)*

b) *SDH Interface*

Figure 2 : *High speed data path interfaces*

Section 4 explains the usage of this high speed core for a large demonstration application. While the resulting capabilities and limitations are discussed in section 5.

High–speed data path interfaces

The system provides two different kinds of data path interfaces. The optical 125 Mbit/sec interface is shown in Figure 2a: after the electrooptical modules standard FDDI components do the serial/parallel conversion to/from a 5 bit wide data path operating at 25 MHz.

The second data path interface for 155.52 Mbit/sec (Fig. 2b) is connected to an SDH (STM–1) framer ([14][15]) that does descrambling/scrambling, parallel/serial conversion, and overhead processing. It is configured by the microcontroller. The extracted payload together with framing signals is then provided/collected to/from two 8 bit wide data paths running at 19.44 MHz. The clock recovery is based on an SAW (surface acoustic wave) filter module.

Microcontroller

The memory of the Intel 80186 device is divided into 3 parts. EPROM memory for the system firmware contains a basic boot kernel including routines for target debugging and system configuration. For semipermanent application software and configuration data for the FPGAs, an EEPROM based memory is added. Up to 512 kbytes RAM as working memory are available. A serial communications controller provides two serial ports.

The microcontroller has to process several tasks:

- configuration and control of the SDH framer (normally at power–up) and reaction to its error conditions
- downloading configurations into and resetting the configurable hardware
- remote debugging on the source level from a host system (via one of the serial ports)
- providing a user interface (via the other serial port), e.g. for accepting new configuration data that is to be written into the FPGAs
- accessing or writing back information from/to the FPGAs via a control interface giving the possibility to do use the microcontroller for slower tasks and off–line processing.

III. TOP–DOWN SYSTEM DESIGN

In the previous chapter the focus has been on the hardware part of the design platform. Now the according top–down design environment is introduced. The goal was to provide a flexible and easy–to–use interface to admit the designer to concentrate on functional system design, by automating the implementation process. For this purpose a lot of tools exist in the market. The Mentor Graphics tools have been chosen because of their integrated environment for different design entry techniques and for different target technologies including printed circuit board design.

The design system mainly consists of the Idea Station tools from Mentor Graphics and Place and Route tools from XILINX. Figure 3 shows the top down design path. Design entry can be textual using a high level description technique. VHDL has been used up to now, but Verilog, ABEL and other similar textual descriptions are possible as well. Another more common

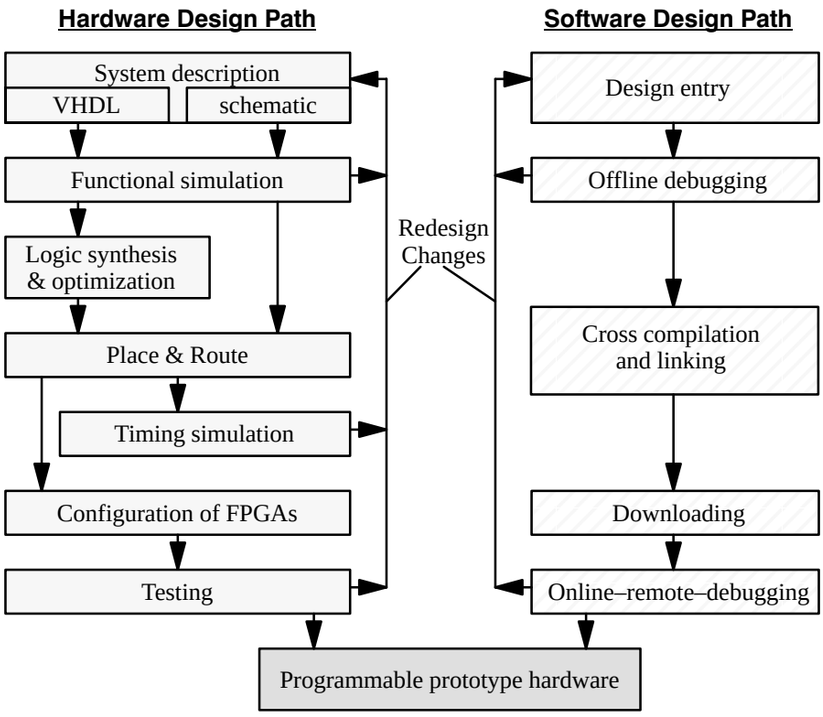


Figure 3 : *Top Down Hardware Design Process*

description technique that is also supported is schematic–based design entry.

After design entry, a functional simulation is necessary to verify the correct logical operation of the desired system. Simulation is supported for functional and schematic–based designs. Designs using a mixture of description techniques can also be simulated. On detection of functional misbehaviors or errors, the system has to be redesigned until the functional simulation performs with the desired functionality.

Once the functional behavior is correct, the next step is the synthesis and optimization of the VHDL descriptions to a gate level description of the system. For this purpose, the Autologic synthesis tool from Mentor Graphics is used. It translates the functional VHDL description to a generic hardware description at the gate level. In a second step it maps these generics to target hardware–specific gates or logic blocks. In this application, a synthesis library for XILINX devices is used that substitutes the generic gates with gates or programmable logic blocks of the FPGA. Different optimization directives for timing and area can be given.

The manually entered schematics together with the automatically generated netlists from the VHDL part are transferred to the XACT place and route software from XILINX. XACT provides automatic placement and routing of the design in the desired target device. This process can be controlled by user directives on placement of system parts and by specification of maximum delays for some signals. XACT determines a placement and routing from these directives and internal optimizations. It outputs a configuration file for the FPGA and back–annotation information for the simulation.

At this step a timing simulation can be done by merging the backannotation containing the expected delays with the design. Now the device can be simulated including switching and propagation delays. At this step normally no errors should occur. However, if in some cases the simulation doesn't perform correctly, a first approach is to refine or extend the timing optimization directives of the place and route tool. The second try is to focus on improvements in the logic synthesis and optimization within Autologic. If this still doesn't lead to an acceptable result, unfortunately there is no other possibility than to redesign the system specification. Our experiences have shown that a functional design, having in mind the implementation technique and possibilities of the XILINX devices normally leads to rather good implementation results.

When the timing simulations perform correctly, the configuration data can be downloaded to the target platform and tested. For test purposes the programmable core of the hardware platform is equipped with several test ports directly fitting a logic analyzer system. This makes it possible to test a lot of functionalities very comfortably.

Apart from this hardware design path, software design for the microcontroller has to be done. Software design and compilation as well as offline debugging can be performed on a standard PC system. A very useful feature of the protocol design platform is the target debugging option. For this purpose, the kernel of the

microcontroller system contains communication mechanisms via the serial interface with the PC based software design system. It further incorporates routines that provide control mechanisms to debug code that is running on the target microcontroller using the normal debugging user interface on the PC remotely

A still missing feature of the the development system is hardware–software–codesign. At the current state of the development tools, no hardware–software cosimulation is provided. Future work may incorporate this useful option.

IV. A DEMONSTRATION DESIGN

A. FDDI–on–SDH repeater application

A FDDI–on–SDH remote repeater for the interconnection of FDDI rings (e.g. at different customer premises) via SDH connections (Fig. 4) has been chosen as the first demonstration design for this platform. Solutions for this kind of interconnections had been proposed and investigated in [1].

This example has been chosen due to its rather complex operations that have to be performed in real time and offer a good possibility to test intensively the hardware platform as well as the software environment. Further the functionality was already well known and proven from a preceding conventional implementation [2] using discrete, standard TTL devices (showing some EMC problems from the rather high clock speeds for this technology).

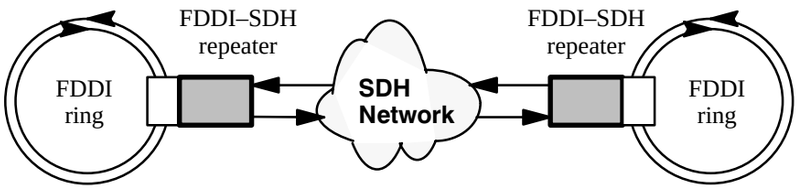


Figure 4 : *Connection of two FDDI rings via an SDH network*

The coupling of the LAN rings (e.g. at different customer premises) is done at the physical FDDI layer by mapping the raw FDDI data stream (4B5B encoded at 125 Mbit/s) into C4 containers within the SDH STM–1 frames corresponding to the so called Super Rate Mapping Protocol [3]. Meanwhile this has also been considered in corresponding FDDI standards [16]. The advantages of the repeater solution over a bridging approach is that no filtering and routing is required, thus achieving optimal throughput and low transmission delays. This is even more important if inter network traffic is dominant and if FDDI rings with small numbers of stations are connected. The latter is typical for today's FDDI networks that are often serving as campus backbones.

Super Rate Mapping Protocol (SRMP)

Within the SDH framing structure (Figure 4) the data stream can be viewed as segmented into rows of

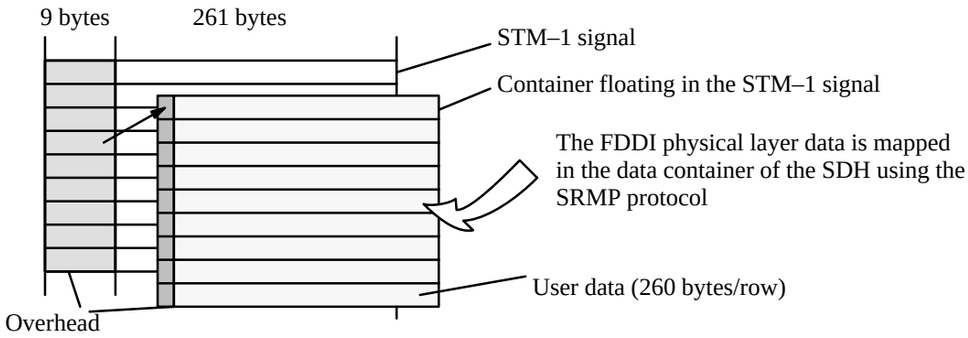


Figure 5 : *STM–1 signal and container (VC4) in SDH*

270 bytes. The first 9 bytes of each row belong to the section overhead. The following 261 bytes form one row of the C4 container. Nine of these rows form an STM–1 frame. The alignment of the container to the overhead is indicated by a pointer (symbolized by the small arrow in Fig. 4) as a result of clock skew. One byte within each row of the container is reserved for the path overhead (the remaining 260 bytes being available for user data).

For the mapping of FDDI data, there are different kinds of bytes defined for the transmission of 0,6,7 or 8 bits of user data. The bitrate of 125 Mbit/s to be transported in the available 149.76 Mbit/s of the SDH containers is achieved by a mapping protocol with a regular pattern of these different kind of bytes. The detailed structure of this mapping can be found in [1] or [3]. The FDDI network and the SDH link are working asynchronously. For both, a slight clock variation

of 50 ppm is allowed. The fine–tuning of the rate adaptation is accomplished by a variable bitstuffing method.

FDDI clock generation on the SRMP receiving side

On the receiving side the FDDI data is extracted from the SDH container and formed to a sequential continuous data stream. A solution for the derivation of the sending clock for the following FDDI link is to use a VCO controlled by the fill level of an elasticity buffer [1]. The main disadvantage of this solution is the complicated implementation. Furthermore, jitter on the FDDI line due to the varying clock may arise. A more stable and easier implementable method has been found in [2]. A simple, fixed frequency oscillator is used to generate the clock for the outgoing FDDI line. The rate adaptation of the incoming SDH–based data stream to the outgoing FDDI data stream is accomplished by the preamble stuffing technique of FDDI. In FDDI data is formatted in packets of variable length. Each packet is preceded by a preamble of nominally 16 symbols (1 symbol=5 bits). Rate adaption in FDDI is achieved by stuffing (insertion, deletion) of preamble symbols. The applicability of this method has already been shown in a conventional implementation of the protocol [2].

B. Implementation

For the implementation on the prototyping platform, the functionalities have been described using VHDL. This allowed an early functional simulation and the reuse of already defined functions. For example the mapping protocol uses the same state machine in the mapping (sending) and demapping (receiving) protocol entity.

FDDI–SDH–Mapper

The mapper has been partioned into 4 entities, namely *map_ctrl*, *fifo*, *shifter* and *stuff_ctrl*. Figure 6 gives an overview of the design.

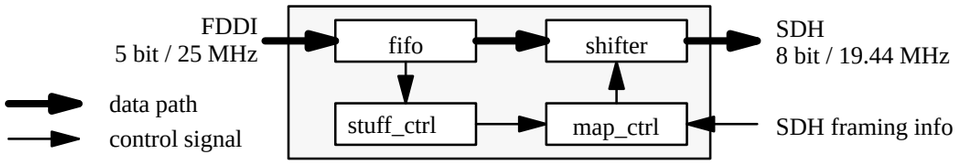


Figure 6 : *Block diagram of the FDDI–SDH–Mapper*

The entity *map_ctrl* is a state machine representing the Super Rate Mapping Protocol. It consists of 3 counters and a subsequent (large) decoder logic. The use of VHDL allowed a very efficient description of the counters as well as the decoder logic. The data arriving from the FDDI network forms a continuous data stream, but the data sent to the SDH network is formatted in containers with overhead. This leads to a discontinuous data stream requiring a fifo buffer to adapt to the continuous stream. The FIFO is built of an array of D–FFs. Read and write pointers are realized using one–hot encoding. The realization of the FIFO had to be rather tricky, because simultaneous read and write access may occur, and read and write clocks are totally asynchronous. Here also the VHDL description technique allowed an early simulation and debugging. The alignment of the user data according to the SRMP as 6,7 or 8 valid bit words has been realized using a barrel shifter. Because up to 15 bits had to be shifted, the shift operation could not be performed within one clock cycle. A multistage barrel shifter has been designed using the VHDL design entry method. The *stuff_ctrl* entity realizes the control of the variable bitstuffing for FDDI–SDH rate adaption in the mapper.

Demapper

The demapper is partitioned into 5 entities. In Figure 7 the corresponding block diagram is shown. For the mapping protocol the *map_ctrl* entity of the mapper could be reused. The rate adaptation from the discontinuous SDH data stream to the continuous FDDI data stream required a fifo buffer. Unfortunately the fifo buffer from the mapper design couldn't be reused, because it already incorporated a part of the 5 bit to 8 bit coding. But a lot of the internal functionalities of the mapper fifo could be reused to build the demapper *fifo*.

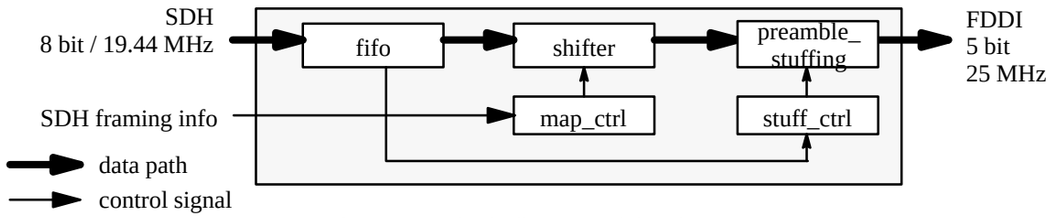


Figure 7 : *Block diagram of the demapper*

The translation of SDH–bytes with 6, 7 or 8 valid data bits to 5 bit FDDI symbols has been provided using a barrel shifter. As already described in the mapper

section, the barrel shifter in the demapper is also realized in a multistage design, because of the high number of possible shift operations. The control of the preamble stuffing has been realized in the *stuff_ctrl* entity. The stuffing of the preamble symbols is executed in the *preamble_stuffing* entity.

For more detailed information on this repeater design [1] and [2] should be consulted. In the following table an extract of the design statistics and device utilization reports for the two FPGA devices is shown. With 86% and 77% CLB (configurable logic block) utilization the devices are near their limits, because routing constraints normally do not allow to occupy all the available CLBs.

MAPPER	No. Used	Max Available	% Used
Occupied CLBs	345	400	86%
Bonded I/O Pins:	36	61	59%
CLB Flip Flops:	323	800	40%
3–State Buffers:	165	880	18%

DEMAPPER	No. Used	Max Available	% Used
Occupied CLBs	311	400	77%
Bonded I/O Pins:	35	61	57%
CLB Flip Flops:	344	800	43%
3–State Buffers:	165	880	18%

V. EXPERIENCES

The use of the high level description language VHDL simplified the design entry to a great extent. It allowed to describe the functionalities very efficiently and made the reuse of entities possible. For example the *map_ctrl* entity of the demonstration example could be used in both the mapper and the demapper system. Parameterized entities (currently not yet supported by the synthesis system) will further increase the reuse of modules. Entities that could not be reused on an as–is basis can contribute to another design by the reuse of their subentities (e.g. *fifo* of the application example) or at least by the redesign of their source code.

VHDL is very similar to high level languages known from software development (PASCAL, C) thus minimizing the learning effort. But the main drawback comes from the operation near the timing and frequency limits of the FPGA target technology. This requires that, in contradiction to the top–down design paradigm, it is necessary to consider the expected synthesis results (in the context of the target technology) already at design entry. Our experience is that the learning of the VHDL language and the introduction to the design tools is rather easy and well supported by tutorials. An unexperienced user needs about 6 weeks for this. However, the consideration of the target technology in the design entry phase is very difficult for the first time user, leading to an increased effort for redesigns.

The simulation tools offer the possibility for an early detection of errors by simulation of single entities. The simulation of the entire design allows to verify the functionality of the new system. At this level it is only possible to verify the functional behaviour of the system. However, for the evaluation of the real performance of the protoype it is also necessary to look at the timing of the design. Due to the relatively high routing delays within the FPGAs the timing cannot be determined before the synthesis and place & route steps have been done. Since they normally require quite an amount of time (more than 3 hours on a SUN Sparc–20) the redesign loop is slowed down. Thus it resulted to be quite advantageous to gain some experience about the target technology to be able to consider the restrictions already at the design entry and thus minimize the number of necessary redesign loops due to timing problems.

The automated top–down design process can accelerate the path from specification to prototype resulting in a faster verification of new ideas and thereby leading to a better time to market. The time necessary for the system development is repartioned by this top–down approach. As shown in Figure 8 the conventional schematic–based design approach requires nearly no training due to the well–known techniques. However, the design and redesign effort is rather large (Fig 8a). With the new top–down development the design and redesign effort can be optimized, but a training phase is necessary especially to consider the target technology. New designs will be easier, due to the well known and well configured development system. In a long term view, the expected design times depend on the number of designs already implemented and the experience gained (as qualitatively shown in the learning curve of Figure 9).

The following table summarizes the expected and the experienced properties of the top down approach as discussed above.

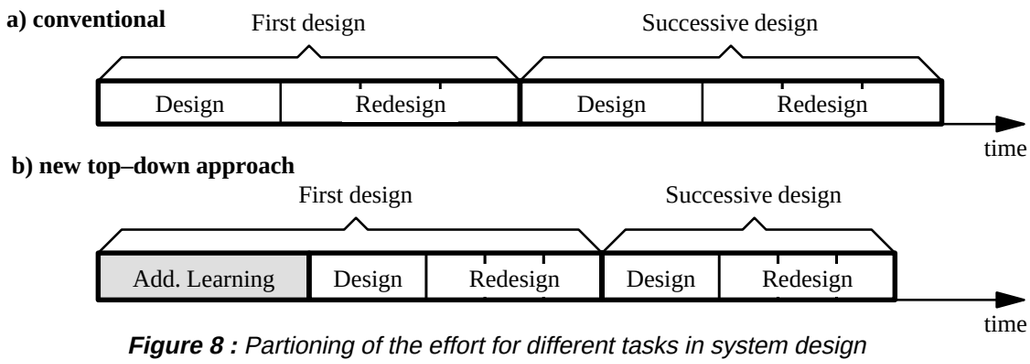


Figure 8 : *Partioning of the effort for different tasks in system design*

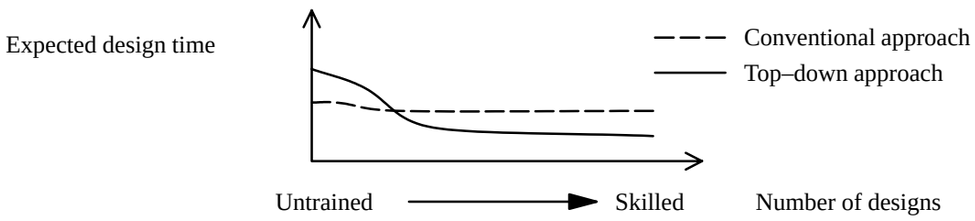


Figure 9 : *Expected design times for long term application of top down methods*

Property	Expected	Experienced
Efficiency of design entry	high	high
Simplicity of redesign	simple	moderate, due to consideration of timing
Reusability of code	good	good
Early simulation	simple and efficient	simple, but since no timing is considered, no statement on implementability
Training	few training necessary	increased training necessary, due to consideration of target technology

In the demonstration example the microcontroller served only for the control of the hardware part, thus the missing cosimulation option wasn't too grave, but for future applications this feature will possibly be necessary. With the provided tools the software design and debugging for the demonstration design (about 1000 lines C source code) could be done in 2 weeks.

VI. SUMMARY

For quick design, verification, and prototyping of communication protocols a development environment has been implemented. The versatile, programmable hardware platform as well as the incorporated design tools have been discussed. A demonstration example together with experiences from its implementation has shown the advantages of the technique. The easy programmability and the well supported downloading options led to quick turn–around times for redesigns. The provided test interfaces offered extensive measurement possibilities and have been used intensively. So the functionality of the implemented system could easily be verified.

To review the advantages of the new development environment the main features are summarized here:

- High–level design entry (VHDL, schematic based, others)
- High–level functional simulation
- Automated implementation process
- Versatile hardware platform allowing fast implementation and redesigns due to its programmable FPGA core
- Effective design environment for control software development
- Universal test and measurement interfaces for comfortable debugging

With all these features the proposed development environment is a valuable tool for the design and evaluation of high speed protocol systems. The main drawback comes from the timing restrictions of the current FPGA technology. Future work mainly has to focus on

the integration of hardware–software codesign concepts. At the moment a second demonstration application (ATM protocol conversion) is planned.

VII. REFERENCES:

[1] A. Kirstädter, J. Weingart, *"FDDI on SONET/SDH Links"*, EFOC, Paris 1992

[2] A. Iselt, *"Implementierung der Mapping und Demapping Einheit eines FDDI–SDH–Repeaters"*, (English title: "Implementation of the Mapping and Demapping Entities of an FDDI–SDH–Repeater"), diploma thesis 1993, Technical University of Munich, chair for communication networks, Prof. J. Eberspächer

[3] E.O. Rigsbee, *"Proposed Super Rate Mapping for ASC X3T9.5 (FDDI)"*,T1X1.5 Optical Hierarchical Interface Group, February 1990

[4] J. Schiller, T. Braun, M. Zitterbart, *"Implementation Architecture for Communication Subsystems"*, 1994 IFIP 4th International Workshop on Protocols for High–Sped Networks", Vancouver, 10.–12.August 1994

[5] J. Schiller, T. Braun, M. Zitterbart, *"Implementation of Transport Protocols Using Parallelism and VLSI Components"*, SBT/IEEE International Telecommunications Symposium 94, Rio de Janeiro, 22.–25.August 1994

[6] J. Schiller, T. Braun, *"VLSI–Implementation Architecture for Parallel Transport Protocols"*, IEEE Workshop on VLSI in Communications, Stanford Sierra Camp, Lake Tahoe, CA, USA, 15.–17. September 1992

[7] M. Kaiserswerth, *"The Parallel Protocol Engine"*, IEEE/ACM Transactions on Communications, vol. 1, no. 6, December 1993

[8] A.S. Krishnakumar, W.C. Fischer, K. Sabnani, *"The Programmable Protocol VLSI Engine (PROVE)"*, ICC 1992, June 1992, Chicago, USA

[9] A.S. Krishnakumar, J.G. Kneuer, A.J. Shaw, *"HIPOD: An Architecture for High–Speed Protocol Implementations"*, 4th IFIP conference on high performance networking, 14.–18.December 1992, Liège, Belgium

[10] D.C. Feldmeier, *"A Framework for Architectural Concepts for High–Speed Communication Systems"*, IEEE JSAC, vol. 11, no. 4, May 1993

[11] D.C. Schmidt, T. Suda, *"Transport System Architecture Services for High–Performance Communications Systems"*, IEEE JSAC, vol. 11, no. 4, May 1993

[12] T.F. La Porta, M. Schwartz, *"Architectures, Features, and Implementation of High–Speed Transport Protocols"*, IEEE Network Magazine, May 1991

[13] Am 79865 and Am 79866 Data Sheets, Advanced Micro Devices Corp., Sunnyvale CA, USA, 1992

[14] SYN155 Data Sheet, TranSwitch Corp., Shelton CT, USA, 1995

[15] SOT–3 Data Sheet, TranSwitch Corp., Shelton CT, USA, 1995

[16] ANSI X3.148, FDDI physical layer protocol specification (PHY)