A Rapid Prototyping Environment for ASIP Validation in Wireless Systems

Matthias Alles, Timo Lehnigk-Emden, Christian Brehm, Norbert Wehn
Microelectronics Systems Design Research Group
University of Kaiserslautern
67663 Kaiserslautern, Germany
Email: {alles, lehnigk, brehm, wehn}@eit.uni-kl.de

Abstract

Current and future wireless devices have to contain flexible outer modems in order to support the many different communication standards used today. Application-specific instruction-set processors (ASIPs) are promising candidates to deliver this flexibility with low power and small silicon area. Their high flexibility allows to support a vast number of different standards. However, testing of such ASIPs is very challenging. Statistical simulations are required, meaning that only after running the channel decoder for many thousand blocks it can be considered to be working correctly. Simulation on nowadays PCs lacks at least three orders of magnitude in throughput. In this paper we present an ASIP prototyping platform, that validates an ASIP in conjunction with its running programs. The ASIP and its baseband processing environment (encoder, modulator, noise channel, etc.) are mapped into an FPGA-based prototyping platform to perform hardware accelerated emulation. Compared to software simulation of the processor pipeline on state-of-the-art computers we could achieve speed improvements of three orders of magnitude.

1 Introduction

Channel coding is the central processing task of the outer modem and is essential for any (wireless) data transmission and data storage. Errors which are induced during transmission or storage, e.g. due to noise, can be corrected on the receiver side with advanced coding techniques. In the past turbo codes and low-density parity-check (LDPC) codes have become very popular, since they belong to the most efficient codes known today. Due to their excellent error correction capabilities they already were adopted in many standards. Table 1 gives an overview of the channel codes used in current standards. Convolutional codes (CC), binary turbo codes (bTC), duo-binary turbo codes (dbTC), and LDPC codes are established channel coding schemes.

Future mobile and wireless communication networks require flexible modem architectures to provide seamless services between these different network standards. Application-specific instruction-set processors (ASIPs) are promising candidates that offer the needed flexibility at low power consumption and small footprint. E.g. in [1][2] the FlexiTreP ASIP family is presented. It supports convolutional, turbo and LDPC codes and thus a vast number of standards, cf. Section 2.

The decoding process of turbo and LDPC codes is done via a heuristic which iteratively works on complete data blocks of considerable length (e.g. 15000 bits for UMTS). Unfortunately, there is no closed mathematical model for these codes to calculate their error correction capability. Hence, to investigate the performance of these codes, Monte Carlo simulations become mandatory. This can be a very time consuming task since billions of bits have to be simulated. E.g. to test for typical bit error rates (BER) of 10^{-7} about 10^9 bits have to be simulated in order to obtain reasonable results. This corresponds to about 10^6 codewords or frames of 1000 bits just for a single signal-to-noise ratio (SNR) and a single configuration. The Monte Carlo simulations are typically carried out on the C/C++ code level.

Validation of channel decoders on the register transfer level (RTL) by performing Monte Carlo simulations is preferable. In this way, system performance relevant errors in the decoders could be detected. This, however, is infeasible due to the poor RTL simulation speed of nowadays PCs. At least a speed improvement of three orders of magnitude is required. Thus, only a few frames are simulated on the RTL level and the results are compared with a bit accurate C model. To overcome this problem, hardware-accelerated emulation can be used. E.g. in [3] a rapid prototyping platform for dedicated channel decoders is introduced. Like this, Monte Carlo simulations for the dedicated cores become feasible and system performance relevant errors are detectable.

In this paper we use hardware-accelerated emulation as well. The immense flexibility offered by the ASIP approach, however, leads to a drastically increased ex-

pense for the validation compared to that of dedicated IP cores. For ASIPs it is not only necessary to verify the processor itself, but also the programs that are running on the ASIP. The ASIP design flow, cf. Section 3, allows to carry out some initial debugging. For instance graphical pipeline debugger and C++ pipeline simulation model exist. Recently, formal verification was presented to verify such an ASIP on the instruction level [6]. But as mentioned before, these techniques are not sufficient in the context of channel coding, e.g. the ASIP programs can contain errors that are only detectable when performing Monte Carlo simulations. The insufficient simulation speed on nowadays PCs is a big obstacle when testing the ASIP for the many use cases, i.e., the different standards, block sizes, and code rates.

To the best of our knowledge there is only one paper about prototyping ASIPs for channel coding. In [4] the authors present an ASIP prototyping platform for turbo coding. This platform, however, is very limited. For each test of a single codeword new synthesis, place and route runs are required. Monte Carlo simulations are prohibited by this approach, since test data is not generated on-the-fly.

In this paper a new rapid prototyping platform for ASIPs is presented. This platform allows to emulate the ASIP on an FPGA, see Section 4. The complete baseband processing environment, i.e., encoder, modulator, noise channel, demodulator and decoder (ASIP), is implemented on this prototyping board. Our results show, that we achieve a speed improvement of three orders of magnitude, compared to conventional simulation on standard 3 GHz PCs, cf. Section 5. This improvement allows to perform Monte Carlo simulations by decoding millions of blocks in minutes and hours instead of days or even months. The platform therefore can be considered as an additional software development platform. It allows us to quickly validate the ASIP programs.

2 ASIP Design Methodologies

ASIPs allow for smooth trade-offs of implementation flexibility (in terms of software programmability) against hardware performance (throughput, energy, area). Application specific instructions offered by these processors can close the performance gap between traditional processors and dedicated hardwired solutions and improve the energy efficiency.

One can distinguish between top-down and bottom-up design methodologies for ASIPs. In the traditional top-down approach, one starts with a generic architectural processor template, e.g. a 5-stage RISC pipeline, see Figure 1a). By profiling the target application on this processor the bottlenecks are identified and removed by inserting application specific instructions. In contrast, the bottom-up approach starts from the micro-architecture level, see Figure 1b). By a thorough analysis of the target algorithms and the required flexibility, functional blocks (triangles) are identified that need to be included into the processor pipeline. All the flexibility and overhead in the processor that is not needed by the application is removed. Adapting bit widths, arithmetics, memory hierarchies etc. completely to the needs of the application yields best performance and energy efficiency. ASIPs designed in this manner share hardly any characteristic of classical RISC pipelines and can be considered as application specific building blocks enhanced with programmability. To emphasize this property these special processors are also called weakly programmable IP (WPIP).

Standard	Codes	States	Rates	Infobits
GSM	CC	16,64	1/4...1/2	...876
EDGE	CC	64	1/4...1/2	...870
UMTS Rel.99	CC	256	1/4...1/2	...504
	bTC	8	1/3	...5114
CDMA2000	CC	256	1/6...1/2	...744
	bTC	8	1/5...1/2	...20730
HSDPA	bTC	8	1/2...3/4	...5114
LTE	bTC	8	1/3	...6144
DAB	CC	64	1/4	none
DVB-H	CC	64	1/2...7/8	1624
DVB-T	CC	64	1/2...7/8	1624
DVB-RCT	dbTC	8	1/2, 3/4	...648
IEEE802.11a/g	CC	64	1/2...3/4	...4095
IEEE802.11n	CC	64	1/2...3/4	...4095
	LDPC	–	1/2...5/6	...1620
IEEE802.16e	CC	64	1/2...5/6	...864
	dbTC	8	1/2...3/4	...4800
	LDPC	–	1/2...5/6	...1920

Table 1 Selection of standards and channel codes

In this paper, we consider the FlexiTreP ASIP family which belongs to this class of weakly programmable IPs. These ASIPs were designed for channel coding (convolutional codes, binary/duo-binary turbo codes, and LDPC codes) and support a vast number of communication standards. Figure 2 shows the fully customized processor pipeline with the functional blocks that are used for turbo decoding. The pipeline employs an elaborated memory infrastructure, that enables to load and store data when needed or computed yielding a high internal memory bandwidth. Functional blocks between turbo and convolutional decoding are shared when possible to increase the architectural efficiency. No general purpose instructions exist. Instead, highly customized instructions provide just enough flexibility to handle the various standards and to reduce the overhead of a flexible processor approach to a minimum. The Dynamically Reconfigurable Channel Code Control holds certain code parameters. This reconfigurability allows to change the data routing within the pipeline depending on the channel code that has to be processed. For further details on the FlexiTreP ASIPs please refer to [1][2].

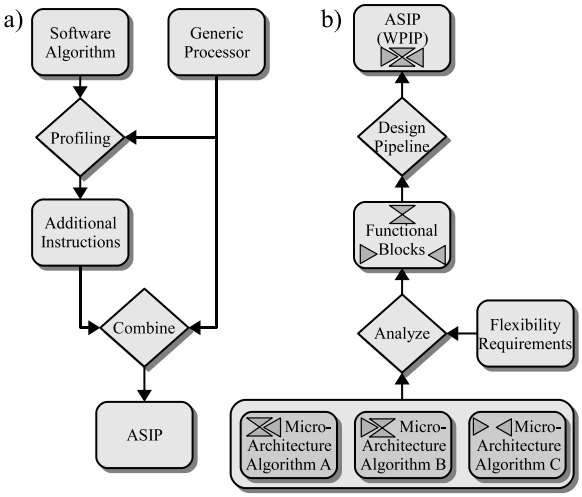


Figure 1 ASIP design methodologies a) top-down approach b) bottom-up approach

In the following section the ASIP design flow is described.

3 ASIP Design Flow

We used CoWare's Processor Designer [5] to implement the FlexiTreP ASIP, since it offers the needed flexibility to follow the bottom-up ASIP design flow as presented in the previous section. Its typical design flow is shown in Figure 3. LISA (Language for Instruction-Set Architectures) is used to model the processor pipeline. According to the flexibility offered by LISA many properties can be defined, e. g.:

- the number of pipeline stages
- the instructions, their syntax, and their binary encoding
- the location of memories within the pipeline
- the operations that are triggered by the supported instructions
- the interface of the ASIP

The Processor Designer allows to generate automatically several tools and models from the LISA description of the pipeline:

- software development tools (Assembler, Linker, Disassembler)
- a cycle accurate simulation model in C++
- a graphical pipeline debugger
- a synthesizable RTL description in VHDL or Verilog

Debugging is performed in two steps: At first the conceptual errors in the model are tackled by simulating the ASIP on the C++ level. Here, the graphical pipeline debugger and the C++ simulation model are used. An executable binary is needed in order to stimulate the processor. In case the pipeline shows erratic behavior, the LISA description or the assembler program have to be adjusted iteratively. When no further errors can be found in the pipeline on the C++ level, the HDL code

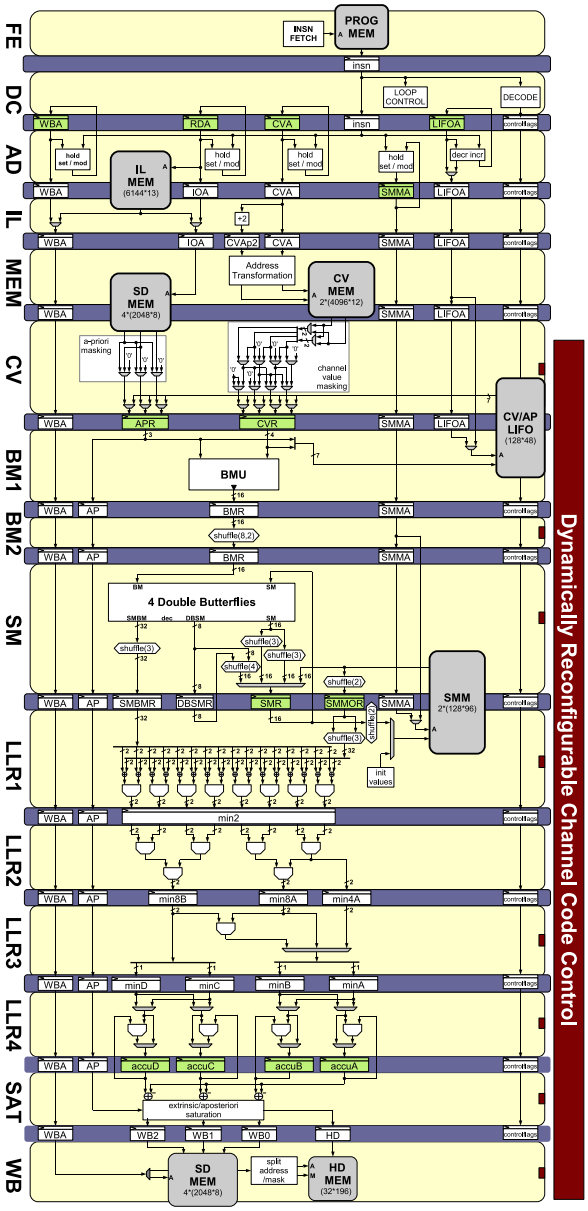


Figure 2 The FlexiTreP Turbo Decoding Pipeline

is generated. In this second step this code is simulated with an HDL simulator while executing a program to make sure that C++ and HDL simulation behave the same.

Using the C++ and RTL models for simulation is a good starting point for testing the correct functionality of the ASIP. Conceptional faults can be found since these models provide access to any signal in the design and any value at any time step can be observed. But several problems might be undiscovered. E.g. range overflows might not occur in the simulated blocks. Furthermore, it is possible that certain sequences of instructions result in an unexpected behavior. This can happen, when there are data dependencies between instructions that have not been considered. Simulating all possible in-

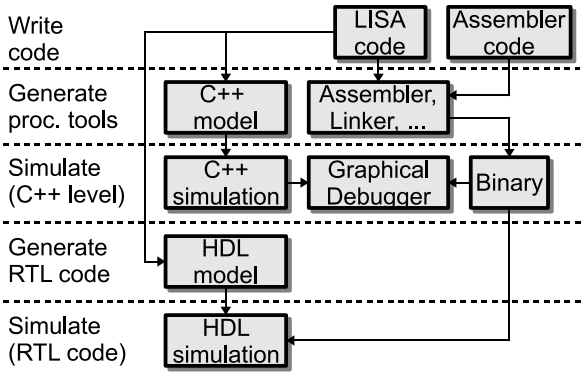


Figure 3 Processor Designer design flow

struction sequences, however, is not feasible.
Formal verification is a important and well established technique to verify RTL blocks. Recently, [6] presented a novel verification technique based on interval property checking for verification of weakly programmable processors, such as the FlexiTreP ASIP family. With this, the correctness of each instruction can be proven by verifying each single operation within the pipeline that is associated with the instruction to verify. Potential range overflows are discovered on this level. Furthermore, constraints of the instruction-set can be extracted. For instance it is possible to say that instruction B must not be executed withing the next n clock cycles after instruction A has been executed. Some of these dependencies might just be bugs, others could be part of the instruction-set architecture, where some instruction sequences are just forbidden.
Formal verification in the context of channel coding, however, cannot supersede excessive Monte Carlo simulations, since it is not possible to verify the whole program that is running on the ASIP. Only after simulating hundreds of thousands of blocks we get the needed frame and bit error rates (FER and BER), that show whether the requirements have been met or not. Typically, frame error rates are in the region of 10^{-4}, meaning that on average only every ten thousandth block contains errors after decoding. Certain errors in the software can only be detected at that error level. Thus, we have to simulate that much blocks with the ASIP for the various block sizes, code rates, and so forth.

4 Rapid Prototyping Platform

To overcome the aforementioned validation bottlenecks, we developed a rapid prototyping environment. We chose the Xilinx ML507 evaluation platform [7] for rapid prototyping the ASIP, since this platform perfectly fits our requirements. It is a fully featured FPGA board with many interfaces and a hardwired IBM PowerPC 440 on the Virtex5 FPGA. The Xilinx Embedded Development Kit 10.1 (EDK) was used to integrate the prototyping platform.

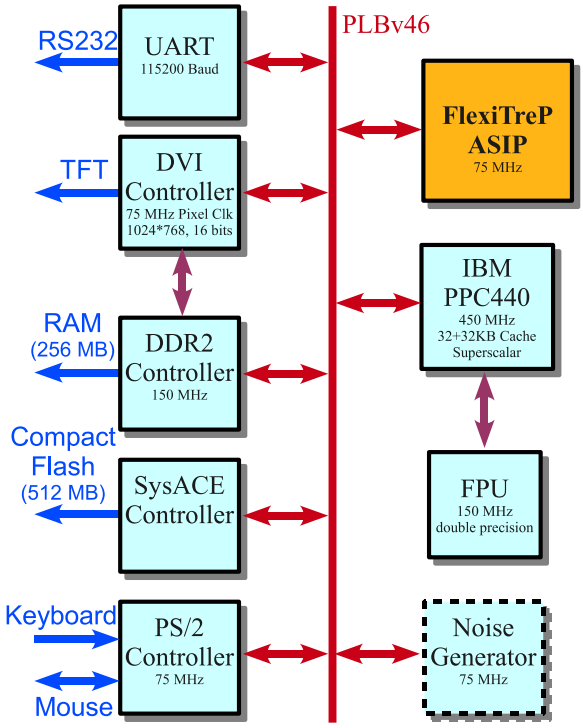


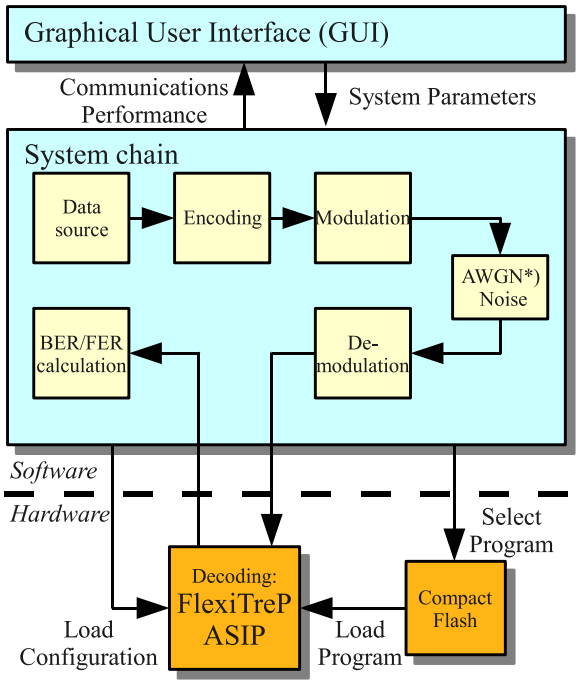
Figure 4 FPGA system overview

Figure 4 gives a coarse overview of the embedded system running on the FPGA. In particular PLB (Processor Local Bus) cores for interfaces (RS232, DVI, DDR2-SDRAM, Compact Flash, PS/2 mouse and keyboard), the PowerPC, an optional noise generator, and the FlexiTreP ASIP are part of the system. A double precision floating point unit (FPU) is available as soft-core.
Prior to running the ASIP three tasks have to be performed:

- A configuration has to be loaded into the ASIP.
- A program has to be loaded into the ASIP.
- Data has to be sent to the ASIP.

Due to the high flexibility of the FlexiTreP, an intelligent interface is needed. Configuration data has to be generated, programs have to be loaded, and input data have to be generated and loaded into the ASIP. Furthermore, FER and BER have to be calculated after decoding. Performing all these tasks in software is the only way to offer the required flexibility. The PowerPC is used for this by communicating with the ASIP via the PLB. For this, a PLB wrapper has been created around the FlexiTreP interface. Configuration and program are loaded in a memory mapped way into the ASIP, while for data transmission handshaking is used.
Figure 5 shows the system data flow, which is based on three components:

- A graphical user interface (GUI) running on the PowerPC [8]
- A baseband processing chain for performing



*) Can also be performed in hardware

Figure 5 Emulation system data flow

BER and FER simulations running on the PowerPC

- The FlexiTreP ASIP running in FPGA logic

The GUI is used to set the system parameters, such as interleaver (e. g., UMTS or LTE), block size, code rate, signal-to-noise ratio, and many more. When all parameters are set, the system chain starts its operation. It consists of a flexible baseband processing model with data generation, encoding, modulation (BPSK, QPSK, 8PSK, 16QAM, 64QAM, or 256QAM), additive white gaussian noise (AWGN), demodulation, and BER/FER calculation. Decoding is performed on the ASIP. The FlexiTreP software driver calculates the required configuration data and loads the selected program from the Compact Flash into the ASIP. After decoding, the results are displayed on the GUI, but can also be sent to a PC, in order to check the results. If these are not satisfying for a certain case, it is possible to use the previously mentioned methods for debugging.

The platform is used as a hardware/software development platform, that allows us to quickly validate the correct functionality of the ASIP in conjunction with the assembler programs. By using the GUI for controlling the whole system, the presented platform is able to run completely autonomously without any PC. Thus, it is suitable for customers of the ASIP that want to develop their own programs. The customers do neither need the LISA source code, nor the C++ pipeline simulation model, nor the VHDL code of the processor, but only the programming model. Figure 6 shows a photo

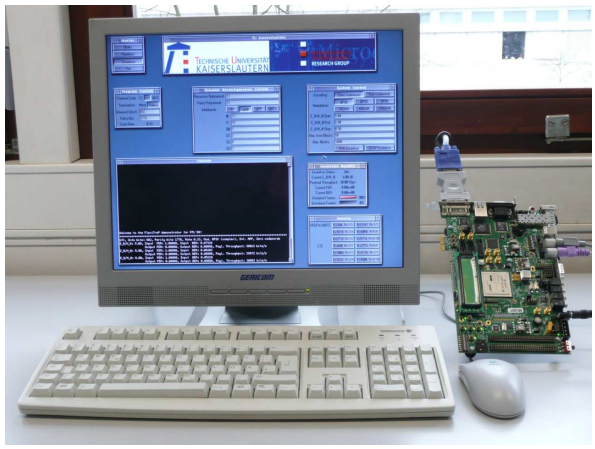


Figure 6 Photo of the rapid prototyping platform

of the prototyping platform.

5 Results

In our setup, the PowerPC is clocked at 450 MHz, the double-precision FPU runs at 150 MHz, and the FlexiTreP ASIP runs at 75 MHz. These clock frequencies were chosen because of the clock ratios that are constrained between all these components. Table 2 shows the resulting payload throughputs for the different validation techniques, i. e., C++ pipeline simulation model, VHDL RTL model, and the presented FPGA-based rapid prototyping platform. For the C++ and RTL simulations a standard PC with a Pentium4 CPU running at 3.2 GHz has been used.

For a UMTS turbo code with 882 information bits, the C++ model achieved a payload throughput of only 450 bits per second, since the ASIP uses mainly non-standard data types, e.g. signed integers with a 6 bit quantization are used for the channel values. With this simulation speed it takes 22 days to simulate one million blocks, which is a necessity for tests at a FER level of 10^{-4}. This simulation corresponds to only a single SNR and a single configuration. The RTL simulation speed of 120 bits per second with ModelSim is another factor of three to four slower.

In contrast to these simulation methods, the prototyping platform achieves payload throughputs of up to 363.3 kbps with a software noise channel. The reason for the large range in throughput are the floating point operations used in the AWGN noise generator. This component takes by far the most computation time on the software side. Changing the code rate or the modulation affects the number of symbols that are transmitted via the channel and thus the number of symbols that have to be distorted by the noise generator. Therefore, for the higher modulation schemes or the higher code rates the number of floating point operations is reduced, resulting in a higher payload throughput. The lowest throughputs for one code rate are achieved when

	Payload Throughput / kbps UMTS turbo code (882 info bits, 5 iterations)			
Code rate	C++ model	VHDL model	FPGA-based (soft-AWGN)	FPGA-based (hard-AWGN)
1/3	0.45	0.12	37.0 - 151.2	217.3 - 252.0
1/2	0.45	0.12	53.7 - 195.3	272.5 - 309.8
2/3	0.45	0.12	71.6 - 267.8	364.7 - 415.0
9/10	0.45	0.12	97.7 - 363.3	507.2 - 578.2

Table 2 Throughput comparison for different evaluation techniques

using BPSK modulation, while the highest ones result from 256QAM modulation.
In order to further increase the throughput, the software noise generator is replaced by a high quality AWGN IP [9]. In this way, the throughput of the overall system can be drastically increased at the cost of flexibility regarding the supported SNRs (the Xilinx IP only supports SNRs in steps of 0.1 dB). Both noise generators rely on the Box-Muller algorithm and have a high precision. Accelerations of up to 600% for BPSK and up to 165% for 256QAM modulation are achievable with the hardware noise channel. Throughputs of more than 575 kbps are possible and the fluctuations in throughput depending on the modulation are reduced. The throughput still varies for different code rates, since execution times of data generating, encoding, modulation, demodulation, and BER/FER calculation are dependant on the bits per codeword.
Figure 7 shows the communications performance of the aforementioned example configuration. These curves were obtained by running the ASIP on the prototyping platform. They are congruent with a turbo decoder software model that applies the same parameters. Up to one million frames have been decoded per SNR with the software noise generator in just two to four hours instead of 22 days. Employing the hardware AWGN channel in this case gains another factor of five in throughput.

6 Conclusions

In this paper we have presented a rapid prototyping platform for ASIPs in the context of channel coding. This platform gives us the opportunity to test the ASIP in conjunction with the programs and to perform FER simulations. Our platform outperforms simulation methods by three orders of magnitude in throughput, what allows to perform Monte Carlo simulations in hours instead of weeks.

Acknowledgment

This work has been supported by the Deutsche Forschungsgemeinschaft (DFG) within the Schwerpunktprogramm “Rekonfigurierbare Rechnersysteme”.

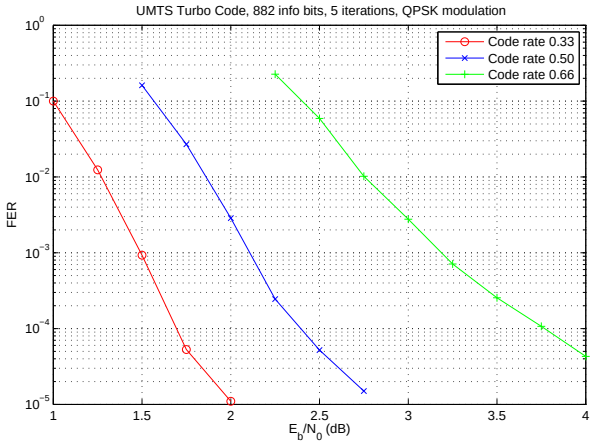


Figure 7 Frame error rates for the ASIP running on the rapid prototyping platform

7 References

[1] Timo Vogt and Norbert Wehn. A Reconfigurable ASIP for Convolutional and Turbo Decoding in a SDR Environment. *IEEE Transactions on Very Large Scale Integration Systems*, pages 1309–1320, October 2008.

[2] Matthias Alles, Timo Vogt, and Norbert Wehn. FlexiChaP: A Reconfigurable ASIP for Convolutional, Turbo, and LDPC Code Decoding. In *Proc. 5th International Symposium on Turbo Codes and Related Topics*, pages 84–89, Lausanne, Switzerland, September 2008.

[3] Martin Irman and Jan Bajcsy. A rapid system prototyping platform for error control coding in optical CDMA networks. *IEEE International Workshop on Rapid System Prototyping*, pages 232–234, June 2005.

[4] Olivier Muller, Amer Baghdadi, and Michel Jezequel. From Application to ASIP-based FPGA Prototype: a Case Study on Turbo Decoding. *IEEE International Workshop on Rapid System Prototyping*, pages 128–134, June 2008.

[5] CoWare. http://www.coware.com.

[6] Sacha Loitz, Markus Wedler, Christian Brehm, Timo Vogt, Norbert Wehn, and Wolfgang Kunz. Proving Functional Correctness of Weakly Programmable IPs - A Case Study with Formal Property Checking. In *IEEE Symposium on Application Specific Processors (SASP 2008)*, pages 48–54, Anaheim, CA, USA, June 2008.

[7] ML507 Evaluation Platform. http://www.xilinx.com/products/devkits/HW-V5-ML507-UNI-G.htm.

[8] Genode FPGA Graphics (FX). http://www.genode-labs.com/products/fpga-graphics.

[9] Xilinx Inc. http://www.xilinx.com/ipcenter.