Implementation Independent Profiling of SDL Specifications

Peter Langendoerfer, Martin Lehmann
IHP
Im Technologiepark 25
15236 Frankfurt (Oder)
Germany
langendoerfer@ihp-microelectromics.com

Abstract. The Specification and Description Language (SDL) is a worldwide accepted standard for development of new protocols, e.g. an increasing number of IEEE protocol standards uses SDL. Especially for mobile devices it is important that the protocol implementation is as energy efficient as possible. This cannot be achieved with a straight forward code generation from SDL to a target language as for example C. Thus, the normal way is that after the specification is done, parts of the protocol are implemented by hand using C or even VHDL. In this paper we are presenting our profiling tool, profSDL, which is designed to support engineers in determining the partitioning of the SDL specification. profSDL allows to determine static effort and dynamic effort of the protocol under development. The dynamic effort can be assigned to the protocol itself as well as to the SDL runtime environment. We applied profSDL to a fully functional SDL model of IEEE802.11a. The design decisions indicated by profSDL matched exactly what the designers decided manually. The major benefit of profSDL is that it works on SDL level, so that the profiling results are independent of any potential implementation decision e.g. programming language features.

1. Introduction

Protocol design and implementation is a complex task. It can be simplified if formal description techniques e.g. SDL are applied. The system under development can be simulated and if it is working correctly, code generation tools can be applied in order to receive a correct implementation. But automatically gained implementations are normally not efficient enough for real world implementations. This is especially true if the protocol is designed for wireless networks since mobile devices are limited with respect to memory, computing power and energy. In order to meet those requirements the protocol development is done twice first using SDL and after the simulation phase the same procedure is done again using a suitable programming language and/or designing hardware accelerators. This slows down the development process and is extremely error prone. The effect of these drawbacks can be reduced if only those parts of the protocol under development, that are time critical, have to be implemented by hand. The open question is which are the computational intensive parts of and how can they be detected as early as possible. Thorough profiling of an implementation can provide useful hints. But the profiling results are then already influ-

enced by the programming language used for the implementation. In addition it may be very difficult to decide to which extend the runtime environment has influenced the overall performance.

In order to tackle these problems we propose to do the performance analysis completely on SDL level. In order to show the feasibility of this approach we have developed a tool, called profSDL, which provides profiling features on SDL level. In addition profSDL allows to differentiate between the computational effort caused by the protocol itself and to the computational effort that stems from the runtime environment.

The rest of this article is structured as follows. First we provide a short overview of related work. Section 3 discusses the benefits of specification level profiling. The analysis features of profSDL as well as its integration in the normal design flow are introduced in section 4. Thereafter we present the results of a case study in which we used a fully functional SDL model of IEEE802.11a. The paper concludes with a short summary of the key results and an outlook on further research steps.

2. Related work

An optimal realization of a protocol especially for devices with scarce resources is very complex task. There are some approaches which intend to provide support for optimizing the architecture of the resulting implementation.

An approach that evaluates different implementation techniques like server model and activity thread model is discussed in [Lang99]. Based on the results the system developer can decide which implementation technique should be applied. The major drawback is that only predefined traces can be evaluated. This means the designer has to know a-priori which parts of the system under development might become a bottleneck.

Profiling on SDL level is shortly discussed in [Li99]. Basically the approach is to identify computational intensive parts of an SDL system by counting how often a transition is executed. This approach is agnostic towards the influence of the run time system. The performance of an SDL based implementation, however, highly depends on the efficiency of the inter process communication which is normally realized by the run time system.

In [Slom00] the authors describe an approach that aims to partition an SDL specification automatically into parts that should be implemented in software and hardware respectively. The basic assumption about the final implementation is that an SDL process is entirely realized in hard- or software. Thus, a fine grained partitioning which maps only parts of an SDL process onto specialized hardware is not supported.

The performance estimation methodology discussed in [Bagh02], provides estimations for the performance of pure hardware, pure software and co-design implementations of a system under development. Here the engineer has to provide an initial starting point for the evaluation phase by defining all alternative architectures.

In contrast to most co-design approaches we do not aim to determine an optimal or nearly optimal architecture for the implementation, but to provide a promising starting point for the design space exploration.

3. Profiling at SDL level

SDL provides an implementation independent description of a system under development. Mapping such a specification on an implementation is a highly complex task. The designer has to decide whether or a code generation tool should be used and for which parts of the system, which actions have to be taken to meet the performance requirements of the system under development etc. The resulting *implementation specification*[1] significantly influences the performance of the system, so that it is needed to find an optimal or nearly optimal architecture. Due to time-2-market constraints developing the implementation specification should be done as fast as possible. And the more parts that can be implemented using code generation tools the better since this saves some time.

The first step during the development of an implementation specification is profiling the system under development in order to get an impression which parts of the system are potential bottlenecks. This is normally done by generating an implementation and profiling it using well known profiling tools. But generating code means that at least the programming language and a certain run time environment (RTE[2]) are already fixed before the profiling starts. So, the profiling data that should help to design an implementation specification is already influenced by an implementation related decision. [daSi00] discusses the performance of Telelogics RTE of the TAU tool, in addition to the remarks in [daSi00] we would like to stress that the signal exchange provided is quite inefficient due to the number of copy operations [Lehm02]. To illustrate the influence of the "programming language" used for profiling we would like give an extremely simplified example. If a procedure executing AES (Advanced Encryption Standard) is mapped into specialized hardware it takes 11 clock cycles to encrypt 128 bit, an optimized software implementation needs about 6800 clock cycles, whereas a straight forward software implementation needs about 22000 clock cycles [Vate03].

We consider the fact that profiling is done on implementation level is as a drawback of the normal design flow. In order to reduce the influence of the prototype-implementation, the profiling should be done completely on SDL level. This can be achieved by counting SDL constructs along possible execution paths [Lang99]. But defining those paths is a lengthy task, at least if a high percentage of the specification should be covered. In addition such an approach also ignores the behavior of the system, i.e. it does not take into account that some parts are executed more often than

[1] The term *implementation specification* denotes a description which defines how a system under development has be realized, i.e. the programming language, soft- and hardware architecture etc.

[2] The RTE for SDL is kind of virtual machine that provides some basic functionality such as timers, signal exchange etc.

others. The dynamic behavior could be taken into account in this approach by defining a load profile. Such a solution would be really agnostic towards implementation issues, but important aspects such as the signal exchange which normally provided by the RTE are neglected, too.

There is definitely a dilemma between absolute independence of prototype implementations and the need to get results quick, easy and for all aspects of the system under development. In addition the influence of the run time environment may not be neglected totally if code generation, at least for a part of the specification, is one of the potential strategies.

In order to remedy this dilemma we propose to use prototype implementations for profiling but with the following restrictions:

- time may no longer be the criterion,
- effort has to be assigned to where it is cause, i.e. it has to be distinguished between effort that stems directly from the SDL specification and effort that is caused by the run time environment.

We propose to use the number of SDL constructs as an criterion to determine the potential effort. There are two different ways how they can be used in order to detect potential bottlenecks. First their number can be counted. This helps to determine where lots of instructions have to be executed. We call this approach static analysis. Second it can be counted how often an SDL statement is executed during a simulation. Thus, a first impression can be gained where the computational effort can be expected. We call this approach dynamic analysis. There might be a significant mismatch between the number of SDL statements gathered during static analysis in one part of the specification compared to the effort that is expected for this par of the specification after the dynamic analysis. This is due to the fact that some parts are only executed when exceptions such as transmission errors have to be handled. A thorough analysis of the results of the static and the dynamic analysis may help to detect such mismatches. They should be analyzed thoroughly. This may help to avoid design errors at an early stage since the analysis of those parts of the system can show that they should be optimized despite that they are executed very seldom.

We also propose to take the run time overhead into account. But also here we vote for a different currency than time. The correct criterion for runtime overhead cannot be defined for all aspects of the RTE. We focus on the inter process communication, since this is where the largest part of the RTE related effort is caused. In addition this is the only part of the runtime effort that can be influence by changing the SDL specification. We use the number of bytes that are copied during signal exchange as metric for the RTE overhead. Reducing copy operations is a suitable means to increase performance. This can be done by merging two SDL processes into a single process or by applying modern implementation techniques like activity-threads [Koen00] or integrated-layer- processing [Twar01].

Abstract performance data, such as those proposed here, indicate potential bottlenecks without being influenced by the performance of the chosen implementation technique. But whether or not a certain part of the specification will become a bottleneck depends highly on the target system (processor, operating system, programming language etc.). In order to allow a performance prediction based on our static and dy-

namic analysis results the execution time of SDL constructs has to be taken into account. So, they have to be measured for a set of potential target systems. In order to predict the performance on a certain target system the profiling data can be customized, by multiplying the profiling results and the time required to execute the corresponding constructs on the target architecture. With this approach a performance prediction for target systems is feasible and in contrast to the traditional approach, it is no longer biased by system characteristics of the development system.

4. profSDL

The major issues during the development of profSDL were to provide the software/protocol engineer with tool support for performance analysis as early as possible in the design flow, and to ensure that the effort needed to complete a certain task can be assigned to its origin, i.e. to respective processes, services etc. or to the runtime environment. In order to achieve these features profSDL supports two profiling phases:

- Static analysis
- Dynamic analysis

The static analysis shows the distribution of coding effort with respect to SDL constructs such as processes, procedures etc. and gives the user a detailed survey of the kind of the used SDL constructs. The dynamic analysis provides information about the computational effort when the protocol is executed. Based on these results a first partition of the system under development can be developed that indicates which parts of the protocol

1. can be implemented by a code generation tool,
2. have to be implemented manually using C or a suitable assembly language,
3. must be realized in hardware.

The profiling results of profSDL are completely independent of the programming language and the target system used. The only exception is the effort needed for signal handling since this is provided by the runtime environment (RTE). In order to minimize the influence of the implementation of the RTE we do not measure time needed for signal exchange but count the number of bytes which are copied between parts of the specification.

In the following subsections we describe the design flow when profSDL is applied, the static and the dynamic analysis as well as their results.

4.1. Profiling with profSDL

In order to evaluate the performance of a system under development it has to be prepared using profSDL. For static analysis it is sufficient to annotate the SDL specification of the system. profSDL extends the SDL specification with a counter for each SDL construct, which is incremented during runtime if the SDL construct is executed.

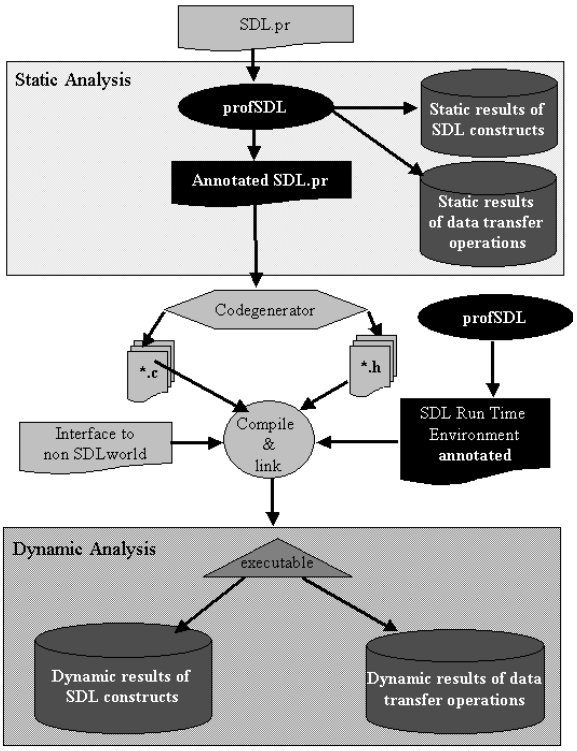

Figure 1: Integration of profSDL into the design flow

While annotating the SDL specification, profSDL generates result files in the static analysis result repositories (Fig. 1). The result files show which SDL constructs are used in which processes, services and procedures. Running the annotated executable delivers the data of the dynamic analysis. Each time an SDL construct is executed its counter is incremented. The results are stored in the dynamic analysis results repositories.

For dynamic analysis it is necessary to annotate also the RTE. This is due to the fact that all data transfer operations are provided by the RTE and that the decision

whether or not data are copied is made while the system is executed. profSDL extends the data handling functions with measurement variables, e.g. for copied bytes, which are incremented during runtime. After finishing the data handling operation, the values of the measurement variables are copied into corresponding variables of the SDL part (process, procedure) which triggered the data handling operation. This ensures that profSDL is capable to provide information which parts of the specification cause which effort for data handling. The effort for data manipulations is recorded in the repository for *dynamic results of the data transfer operations* (Fig. 1).

4.2. Static Analysis

The static analysis is used to gather information about the type of SDL constructs[3], their number and their distribution into parts of the system under development. While collecting these data profSDL annotates the SDL constructs in such a way that all constructs can be counted during the dynamic analysis. During the static analysis we distinguish between "simple" SDL constructs and data operations such as sending signals or assignment operations. This is due to the facts that the effort caused by data operations depends on the data type[4] and that the effort stems from the RTE. This is also the reason why the RTE has to be annotated for the dynamic analysis (see next subsection).

The whole analysis is done automatically and delivers two result files one for simple SDL constructs and a second one for the data operations. The effort caused by the simple SDL constructs is constant, i.e. it does not change during runtime, so the static analysis indicates the effort of a certain section (see Fig. 1). But, the effort resulting from data operations depends highly from the data type. There are SDL data types which are always copied and some which may be passed by copy or by reference. The effort for copy-only data types is constant for each execution of the assignment statement, so counting the related assignment statements provides an indication about the resulting effort. For all other data types the effort caused depends on how the data exchange is realized, and on the amount of data transferred. Both parameters may change over time, i.e. from execution to execution. Thus, a dynamic analysis is needed to determine the effort for these operations.

The result of the static analysis is a file that shows which SDL constructs are used in which processes, services and procedures. In addition it displays also how many constructs of a certain type are contained in a process, service etc.

---

[3] ProfSDL can cope with nested SDL operations, e.g. output, decision etc. , so that it provides a fine grained analysis result.

[4] Simple SDL data types such as integer, Boolean, character etc. are always copied whereas all SDL strings, e.g. SDL_Charstring, SDL_Bit string can be passed by reference or have to be copied.

4.3. Dynamic Analysis

The static analysis indicates how much effort is caused in a certain part of the specification if it is executed, but the open point is how often is a certain part of the specification executed. This question is answered using the dynamic analysis. Here it is counted how often a certain operation is executed and how many bytes are transferred.

Dynamic analysis of simple SDL constructs From the static analysis each construct is already annotated with a counter. This counter is incremented during the dynamic analysis, each time the construct is executed. Thus, the result of analysis indicates where the highest computational effort is located, for the investigated load profile. Those parts of the specification which are very often executed under the expected load are candidates for manual implementation.

Dynamic analysis of data handling operations (runtime_effort) Since several data types may be passed by reference or have to be copied, it is necessary to use an annotated version of the runtime environment. Using the annotated RTE, we can distinguish how data are passed and determine how many bytes have been passed. The effort caused depends on these parameters, e.g. if the data are passed by reference the effort is independent of the datatype and its size. In order to allow a detailed analysis profSDL logs:

1. where the operation was defined (name of structural object (process, procedure etc.)
2. what was transferred (SDL datatype)
3. how often it was transferred (number operations)
4. the type of the assignment operation (copy or reference passing)

The result helps to determine how the runtime environment influences the performance of the protocol under development. It also indicates where the communication inside the protocol can be or should optimized.

5. Case Study: IEEE 802.11a MAC Layer

In order to verify the profSDL approach we used a fully functional SDL model of the IEEE 802.11a MAC layer. This model was developed and implemented at our institute. Thus, we could check whether the results delivered by profSDL indicate the right parts to optimize. For our simulation set-up we instantiated two communicating mobile stations. The mobile stations exchanged data packets, only. The results presented here show the effort at one of these stations.

5.1 IEEE 802.11a SDL specification

The IEEE 802.11a standard specifies the medium access control (MAC) and physical (PHY) layers for a wireless LAN. The standard addresses the problems caused by the properties of the wireless medium by introducing a new medium access control scheme and encryption to secure communication between stations, to name but a few.

The 802.11a MAC layer provides functionality to allow reliable data delivery for the upper layers over the wireless medium. The data delivery itself is based on an a-synchronous, best- effort, connectionless delivery of MAC layer data. There is no guarantee that the frames will be delivered successfully.

The full SDL model consists of an OFDM-Phy block, an LLC block a distributor block and a MAC block. The latter represents the core of the SDL model and is the only part of the specification we consider in this case study. It is subdivided into three

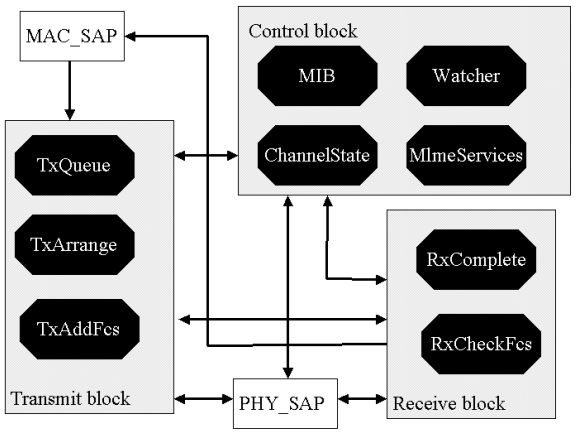

Figure 2: Structure of the SDL model of the IEEE 802.11a MAC layer

blocks Transmit, Receive, and Control, which themselves contain up to four processes

(see Fig. 2). The Transmit block contains functionality for maintaining a queue of frames to be transmitted, initiating their transmission and generating the frame check sequence (FCS). Within the Receive block, incoming frames are checked for correct FCS and the content of the frame is analyzed. The Control block controls the operation of the Transmit and Receive blocks and processes control requests from higher protocol layers.

Table 1. Mapping of the MAC layer functionality onto hardware accelerators, manually written C/Assembly code and automatically generated code

Mac Layer functionality	Hardware Acc.	Hand coded
RxcheckFCS	x	
RxComplete		x
MIB		x
MlmeServices		x
ChannelState	x	
TxAddFCS (CRC)	x	
TxQueue		x
TxArrange		x

5.2. IEEE 802.11 Implementation

The first step during the implementation MAC layer was to use the TAU Cadvanced code generator [Tele03], to get an executable implementation. But, the performance measurements showed that thorough optimizations were needed to fulfill the timing requirements of the 802.11a specification. In general, byte stream processing of frame data (CRC calculation, RC4), timers, back-off procedure, and the logical carrier sense mechanism have been mapped to hardware. All other control and management functions are implemented in software, thereby keeping the flexibility to make changes with little effort, and to better debug and test the system. Table 1 depicts which parts of the MAC block were realized as hardware accelerators or are manually implemented in software.

5.3 profSDL results

Figure 3 shows the result of the static analysis of the IEEE 802.11a MAC layer. The majority of *SDL constructs* and especially of *SDL tasks* is to be found in the SDL pro-

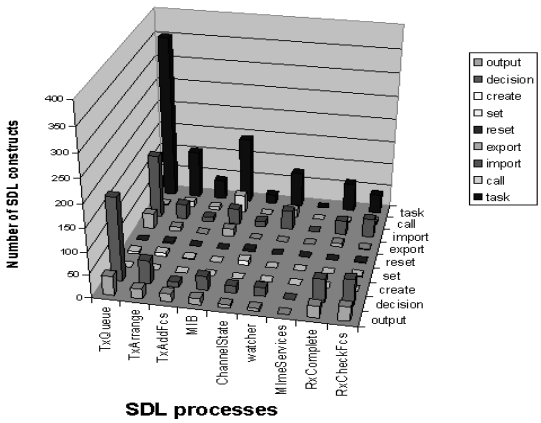

Figure 3: Results of the static analysis of IEEE802.11a MAC

cess `TxQueue`. Whereas in the processes `ChannelState` and `TxAddFcs` only a moderate number of SDL constructs is specified. In an optimization approach which uses only the static data the process `TxQueue` would be the most likely candidate for implementation by hand or realization in hardware. But the results of the dynamic analysis indicate that the effort during the execution of the MAC protocol stems from the processes `ChannelState, TxAddFcs and RxCheckFcs`. Using the results provided by profSDL as a starting point for designing an implementation specification is a promising approach. During the implementation of the IEEE 802.11a exactly these three processes have identified as the computationally most expensive. After thorough analysis they have been realized in hardware, whereas all other SDL processes have been implemented by hand (see table 1).

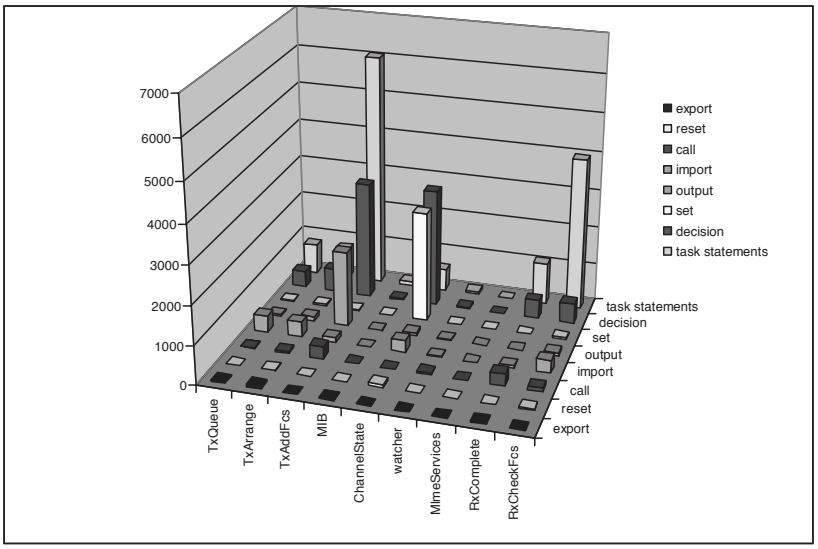

Figure 4: Results of the dynamic analysis of IEEE802.11a MAC

7. Conclusions and Outlook

In this paper we have introduced the concept of performance analysis on specifications level. The benefits of this approach are the independence of implementation issues and that the analysis can be started in very early design phases. In addition our approach allows to distinguish between effort that stems from the RTE and effort that

is caused by the system under development. We have discussed profSDL which implements prototypically our approach. With the MAY layer of IEEE 802.11a we used a highly complex and industrial relevant protocol to test the applicability of profSDL. The result was very promising. The analysis results of profSDL indicated the same partitioning into soft- and hardware as the one chosen by the designers during the implementation of the protocol.

We intend to include additional functionality so that profSDL becomes capable to calculate the execution time after the dynamic analysis. This can be done by providing data about the execution speed for several SDL constructs for a set of target architecture.

References

[Bagh02] A. Baghdadi, N.-E. Zergainoh, W.O. Cesario, A.A. Jerraya, *Combining a Performance Estimation Methodology with a Hardware/Software Codesign Flow Supporting Multiprocessor Systems*, IEEE Transactions on Software Engineering, Vol. 28, Nr. 9, September 2002.

[daSi00] J. L. da Silva Jr., M. Sgroi, F. De Bernardinis, S.F. Li, A. Sangiovanni-Vincentelli, J. Rabaey, *Wireless Protocols Design: Challenges and Opportunities*, CODES - San Diego, US (May 3-5, 2000).

[Koen00] H. König, P. Langendörfer, H. Krumm: Improving the efficiency of automated protocol implementations using a configurable FDT compiler. Computer Communications (Elsevier), Vol. 23 (12), 2000.

[Lang99] P. Langendörfer, Th. Krüger, H. König: *Leistungsbewertung von SDL-Spezifikationen - Ein Werkzeug zur Bewertung von Implementierungstechniken*. Formale Beschreibungstechniken für verteilte Systeme, 10. GI/ITG - Fachgespräch, Shaker Verlag, 2000.

[Lehm02] M. Lehmann, Hardware-unabhängige Evaluierung der Leistungsparameter eines *Performance Evaluierung einer SDL TCP Modells,* , Bachelor Thesis, BTU Cottbus, 2002

[Li99] J. J. Li, S. London, P. Vilela, and J. R. Horgan, *xSUDS-SDL: A Tool For Diagnosis and Understanding Software Specifications*. In Proceedings of 10th International Symposium on Software reliability Engineering, IEEE Press, 1999.

[Slom00] F. Slomka,; M. Dörfel, R. Münzenberger, R. Hofmann: *Hardware/Software Codesign and Rapid Prototyping of Embedded Systems* . In: IEEE Design & Test of Computers (17)2 (2000), S. 28-38

[Tele03] Telelogic, AB,*Telelogic TAU SDL Suite*, 2003

[Twar01] S. Twarok, P. Langendörfer, H. König: *Automated Derivation of ILP Implementations from SDL Specifications*. In Proceedings of the 21st IFIP WG 6.1 International Conference on Formal Description Techniques for Networked and Distributed Systems, Kluwer, 2001.

[Vate03] F. Vater *Evaluierung von Hardware- und Software-Implementierungn eines symmetrischen Verschlüsselungsverfahren (AES) zur Anwendung auf mobilen Endgeräten*, Bachelor Thesis, BTU Cottbus, 2003.