SYSTEM LEVEL HARDWARE ABSTRACTION FOR SOFTWARE DEFINED RADIOS

Cyprian Grassmann (Infineon Technologies AG, CPR ST, 81739 Munich, Germany, cyprian.grassmann@infineon.com), Mirko Sauermann (mirko.sauermann@infineon.com), Hans-Martin Bluethgen (hans-martin.bluethgen@infineon.com), Ulrich Ramacher (ulrich.ramacher@infineon.com)

ABSTRACT

Next generation mobile devices employ multiple programmable processing resources, which need to be orchestrated efficiently by the designer. The usage of low level operating system APIs limits the reusability of the implementation substantially. Moreover, expensive re-implementations are needed during design space explorations. An explicit and platform independent representation of parallelism within the system model on different levels of abstraction is essential for a successful and quick design process as it relies on code generation and compilation techniques. Throughout this paper we discuss the prerequisites for a substantial support of the design process by code generation and compilation techniques and the implementation of according tool extensions. The development of the tool extensions is based on the Eclipse framework [1] and simplified parts from a WLAN 802.11b model are used as test examples for the design process.

1. INTRODUCTION

Increasing complexity of software implemented radio standards and the need for performance enhancements from release to release, fuels the interest in hardware abstraction for signal processing systems and for software defined radios in particular. Normally hardware is abstracted through application programming interfaces (APIs), like they are provided by operating systems. This kind of abstraction specifies an API composed of a common set of services, which are fairly low level. As the API warrants properties of its services, the hardware is already specified to a certain degree. The mapping process is simpler for programs using such APIs but unfortunately it restricts the possible alternatives of hardware architectures to a smaller set. Especially it is nearly impossible to find a common set of services, which abstracts from different kinds of parallel architectures. Similar problems appear with reconfigurable logic. One way to tackle with that problem is the introduction of rich, application specific APIs, which need to be continuously adapted to evolving standards and progresses in the signal processing. Not only the maintenance and porting of such libraries to different architectures, but also the limited reusability of the code, which uses such APIs is the major drawback of that solution. Instead we propose a system level description for signal processing systems, which serves as the origin of a model based design process. Rather than implementing the base band processing in some of the programming languages, using a given OS-API, a functional component view is used, which is extended by so called "non-functional" information, like timing requirements and quality of service constraints. As a functional component view just covers the coarse grain data flow nature of the signal processing system, additional thoughts need to go into the action semantic used inside of the functional components and into the specification of control. Any procedural language like C or Java may be a straight forward solution to describe the action semantic of the functional components, but the mapping to parallel architectures is complicated or prohibited, because data dependencies can't be analyzed completely. Additionally the programmer may be able to give hints for different parallel mappings of the same algorithm, to increase the chance for an efficient mapping to heterogeneous parallel hardware architectures. Also a lot of care has to be taken for a clear separation of control and signal processing. Only if that separation is maintained, efficient partitioning and code generation mechanisms can be developed. An efficient code generation is a key technology for quick design iterations based on the system model that we propose. With this paper we will give a brief overview over the possible candidates for a system description and describe our approach towards a model based design process, which supports the mapping to parallel architectures. As this approach can't be realized without low level APIs, supporting the code generation and compile process, we also discuss the impact of this approach on this kind of APIs. A simplified receive chain from our WLAN 802.11b model serves as concrete example.

2. MODEL BASED SYSTEM DESIGN

The main objectives of a model based design approach are the clear separation of the system modeling and the system implementation onto target hardware architectures and the ability to execute the model for early verification. The separation of modeling and implementation basically allows the reuse of a model with different target hardware architectures, hence it also allows to design the target hardware architecture concurrently to the model.

Nowadays design processes following this concept just partly. Especially for systems which are dominated by signal processing tasks, more or less larger parts of the overall system are modeled using tools like Simulink. Simulinks representation is well suited for data flow models and does not necessarily requires to model the system in a target specific way. Nevertheless a lot of designs already contain a lot of target specific aspects, which prohibit the reuse of these designs in a new system model. It is important to note, that the target specific information itself is not the problem, but rather its intended or unintended mangling with the description of the model behavior. The target specific information should rather be clearly separated as a set of constraints to the model – one set for each target hardware architecture. Modeling only parts of the system obviously prohibits early system verification.

Not only the time pressure during product designs but also the time consuming simulation of the complex system models results in a limitation to partial models of a system. A substantial relieve can be expected by the seamless integration of sophisticated simulation techniques into the design process. Acceleration need to be realized by distributed computing or hardware acceleration, if a further abstraction of the model is prohibited. It should not be unmentioned here, that tools like Simulink also limit the possible design space, as they provide limited semantic and usually focus on a specific computational model like a data flow model. That would lead to an unreasonably high effort for the modeling if systems with a mixture of computational models have to be modeled – which is the normal case and not the exception. Hence the gap between theory and practice is also due to the lack of efficient tool support, which is based on the lack of according techniques to transform the higher level representations of the model to a representation which allows for implementation. As the higher level representation needs to be translated into a representation closer to the implementation, preserving the semantics, it can be referred to as a compilation. Often the term code generation is used for this transformation as a lot of code which will be compiled by compilers of the target architecture is generated from the abstract models of the higher level representation.

Ideally a high-level representation gives the designer sufficient semantically expressiveness and guidance for a description which is independent form the target architecture. Additionally it should be possible to add a rich set of constraints to enable a tool based transformation of the higher level representation to a representation closer to the implementation. It is self explanatory, that the high level representation is executable to allow an early verification.

3. MODELLING LANGUAGES

Modelling languages like the waveform description language WDL [2] are aiming to provide an ideal high-level representation, avoiding limitations by the combination of aspects from different existing languages. WDL suppose to incorporate aspects from functional, object oriented, block diagram, state machine, synchronous and specification languages to form a language for a hierarchical decomposition of behaviour. The refinement to an implementation should take place with help of a set of constraints which is not further detailed.

WDL should be realized using Ptolemy II from University of California at Berkeley [3], as it already fulfils many of the requirements asked for by the WDL specification and allows the execution of the model by a Java simulation.

The question is, why new modelling languages like WDL are invented, even if there is a bunch of languages available to model systems and moreover why the success of all these languages is reasonably low – except for languages in specific domains, like UML in the object oriented domain. On the one hand this is due to a rather conservative attitude of the EDA industry and on the other hand based on the lack of sufficient tool support for the new language capabilities. This is also true for WDL: A central aspect, the refinement to the implementation, including the code generation is still in its infancy. Only if a decent support for these steps in the design process is available, it is reasonable for a system designer to consider the shift to a new modelling language. Additionally legacy code and models are good reasons to stick with an already existing modelling language.

Hence a deeper and longer lasting impact to the overall design process can be expected by focusing onto enhancements to the current state of the art modelling techniques and languages picking up the designer at his current level of abstraction. Additionally a common acceptance and understanding of the modelling language is crucial for its success. Therefore the adaptation and extension of UML [4], like it is actually taking place with SysML [5] will hopefully lead to a widely accepted solution in the community. In addition to this, modelling tools and techniques like Simulink should be interfaced and extended for a continuously migration towards a unified description. The most valuable extension to these tools will be a well defined format and method to add constraints for a refinement to implementation, like with a logic language used in Metropolis [6]. This will lead to a relieve of the high

level description from properties which are specific to the target hardware architecture and will allow for a further automation of the code generation process.
A critical part of each modelling language is the underlying language which is used to describe the behaviour of functional blocks, as this language has to be compiled or synthesised efficiently to the target hardware architecture. Most often programming languages like C, C++ or Java are used. Hence the exploitation of parallelism on this level need to be done by loop nest analysis and exploitation of instruction level parallelism during the compilation. Parallel programming languages can ease the compile process and force the designer to explicitly express the parallelism on this level too. Nevertheless none of the parallel programming languages has come to acceptance so far, because of the low maturity of the compilation techniques and low portion of massively parallel commercial systems in the past. However, due to the raising complexity and parallelism of systems today, explicit parallel programming languages and the explicit documentation of data and control dependencies within the system gets more important.

4. CODE GENERATION

The code generation process and its support by the modelling language needs to be further specified, as code generation can take place on multiple abstraction levels and can potentially cover very different aspects of the design. Ideally it would be possible to describe the behaviour and the structure of the system in very abstract manner, using functional component diagrams and a semantically rich mathematical description for the behaviour of the components. The code generation should then not only be able to analyze the behaviour of each component but rather be able to extend this analysis over multiple components and to modify and optimize the underlying algorithms for an efficient implementation onto the target hardware architecture, including the partitioning into software and hardware implementations.
Such a scenario is currently far of reach, as the code generation engine would need to be able to rework the underlying algorithms of the model without changing the indented behaviour of the system. Up to now this task can only be addressed by the system- and algorithm-designers. Approaches for an automatic derivation of special instructions or dedicated hardware components during the refinement to an implementation are addressed by projects like MOVE [7, 8] and will not be further discussed in this article, as it focuses on a software implementation alone.
Nevertheless code generation can be efficiently exploited to map the structure and control of a system model to an implementation onto the target hardware architecture, taking the behavioural components as black boxes. Consequently the granularity of the model is predetermined by the functional partitioning and the design of the functional components. The main steps in that kind of code generation are to find a valid schedule of the component functions and to declare and link the data and control buffers needed for its execution. This can be done fairly easy for a single task environment, but as soon as multithreading or parallel hardware should be exploited the code generation is getting more demanding, and also more important for a successful and quick design process. Even if there are no satisfying automatisms to find a good partitioning onto parallel hardware targets - which may also be of heterogeneous nature - the design process is still substantially accelerated if all the buffers, synchronization primitives and their arrangement in a parallel program (e.g. by the generation of multiple thread functions) are generated automatically. It allows a quick assessment of different alternatives for a partitioning of the system onto parallel target hardware architectures and avoids error prune, manual modifications of the software implementation which can easily consume person months to run free of errors.
We use a template based code generation process, which allows us to generate multithreaded C-code for simplified receive and transmit chains of WLAN 802.11b. Extensions of the code generator needed for more complex models, including complete receive and transmit chains, are currently under development.

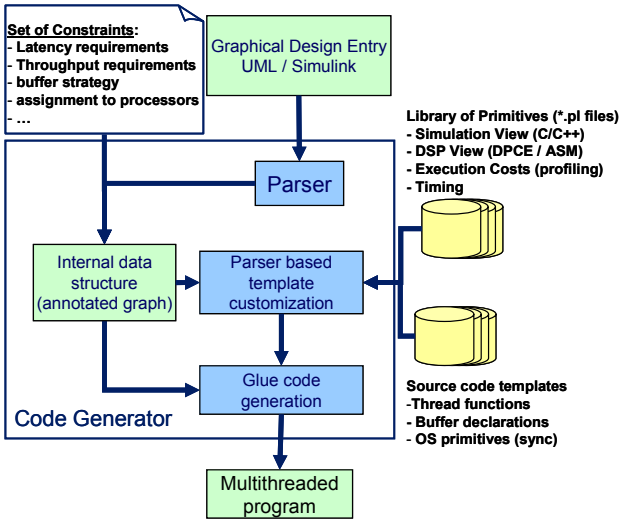

Figure 1: Code generation process

The basic code generation process is shown in Figure 1. The structure of the model is extracted from the Simulink MDL-file. Additionally the data types and sizes of the connections between the functional components are taken from the <block name>.pl files. The pl-file format is based on the ptlang format from Ptolemy. We use this format because it allows us to describe the functional blocks in a fairly generic way, without too many tool specific aspects, contrary to the description with S-functions in Simulink.

To keep the link to Simulink we implemented a generator, which generates S-functions from pl-files.

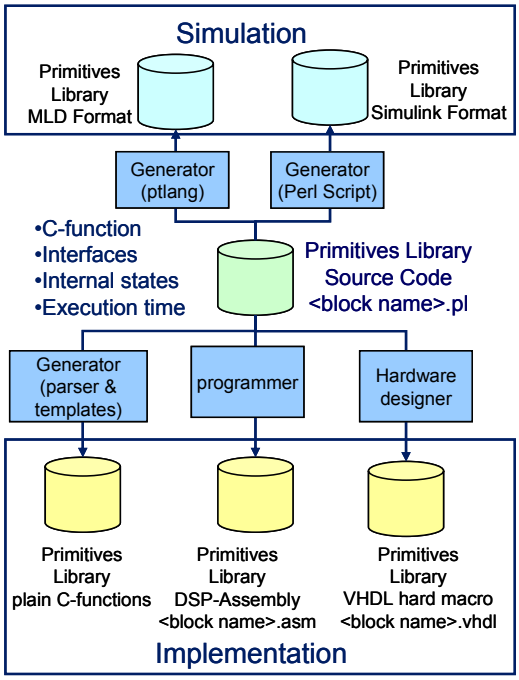

Figure 2: generic representation of functionality

The pl-file format itself is further extended by sections which allow the description of internal states and estimated cycle times for the execution of the enclosed function. Hence it serves as the central generic description of functionality (Figure 2). Additionally we investigate the possibilities to use parallel programming languages and language extensions, which provide a substantial support to the compilation process. In this context we used Data Parallel C-Extensions (DPCE) [9] as input format for our SIMD-compiler which targets a special SIMD DSP-architecture [10]. DPCE appears to be helpful, as a part of the loop nest analysis can be omitted and the generation of sequential code from the parallel code is relatively straight forward. Therefore simulations can take place on the generated sequential code. Nevertheless we also discovered the need for semantic extensions to enable further optimizations within the compiler.

Due to the ongoing research and development of the code generation process the designer currently has to determine the buffer strategy and other "non-functional" aspects to enable the code generation process. Up to now the buffer strategy and similar "non-functional" information is either kept in additional text files or in separate objects within the implementation. We investigate formal and modular formats, like the logical language used in Metropolis, which ensures better portability and consistency of system model.

5. PARTITIONING AND SCHEDULING

Even if the partitioning and scheduling of an application onto the target system is done manually, simulation runs are still needed to assess the quality of the respective solution. To avoid multiple simulation runs with different partitionings and different schedules, an optimization based on the costs for the execution of each functional block and the cost to communicate the results to succeeding blocks can be used to determine the quality and the best candidate from multiple partitionings and schedules. We model each operation mode of the base band processing by a graph, apply a multi level partitioning to it and assess all possible schedules for the given partitioning [11].

For some systems, like for WLAN 802.11b it is necessary to setup multiple instances of the same processing chain to fulfil throughput requirements. It is important to know if one of the functions in the chain has a state which needs to be preserved from one execution of the function to the next, because this state need to be communicated and synchronized between the multiple instances of the processing chain. Therefore we extract state information from the pl-files of the model and use it when multiple instantiations of processing chains leads to an expansion of the original graph. The complete process of the partitioning and scheduling solution is sketched in Figure 3.

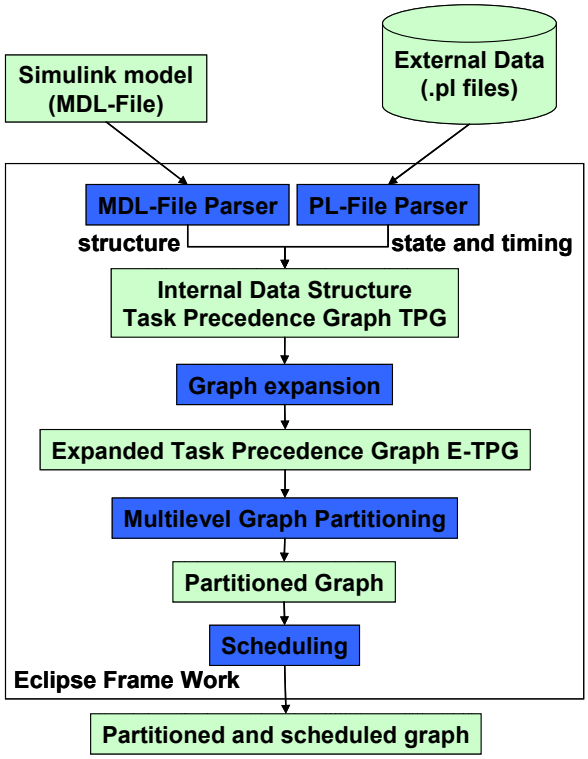

Figure 3: partitioning and scheduling

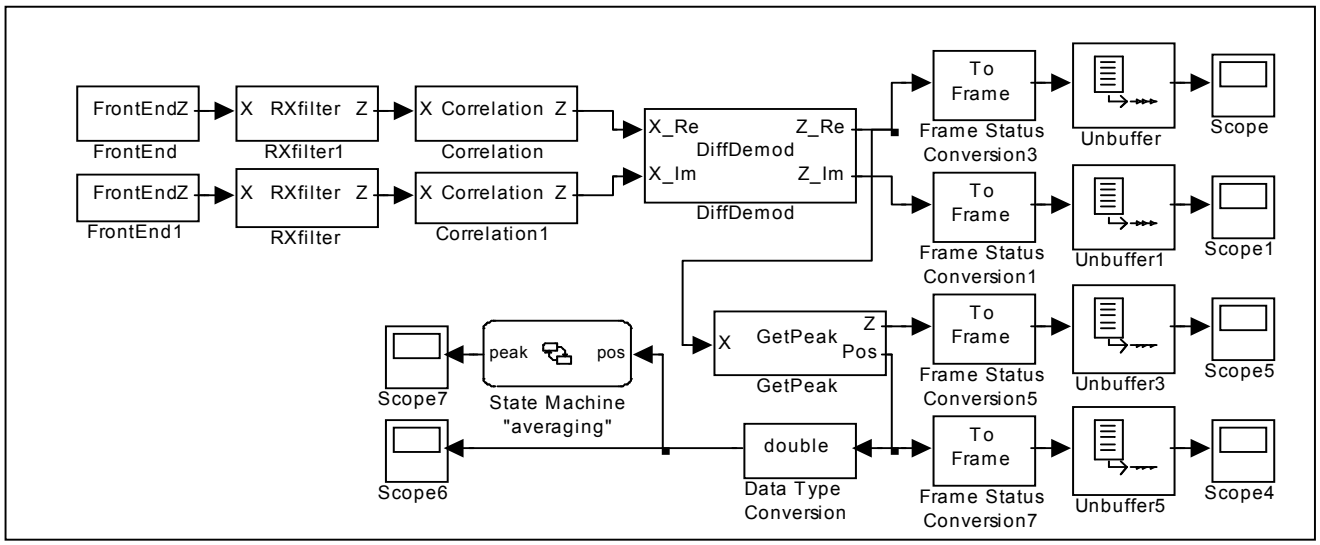

Figure 4: Simulink model of simplified 802.11b receive chain

6. OPERATING SYSTEM

Only a very lean operating system API is needed to support the implementation of wireless base band standards. Hence we developed an operating system API which is reduced to 21 functions, including thread handling, messaging and synchronization. Interrupt handling is optional and can be added if needed. The resulting operating system not only has a very small memory footprint of about 13 kB, but is also highly portable and configurable, because it employs a clear separation of platform dependent and independent parts. All services are implemented platform independent if possible. Based on the minimization and the clear separation of platform dependent and independent parts, we are able to generate the platform dependent parts of the operating system with the help of an architecture description.

7. WLAN 802.11B IMPLEMENTATION

The development of the tool extensions described throughout this paper is guided by a reference implementation of WLAN 802.11b. Hence generated code can easily be checked against the manual implementation. Moreover we gained valuable experience by the manual implementation of the transmit and receive chains for this standard, to introduce improvements at the right places within the design and development process. E.g. the observed relation between signal processing code size and control code size, which is actually 2000 lines versus 8000 lines demands for a code generation solution, which relives the designer from the manual implementation of the control code. This is especially true, because large parts of the code are very regular and with every change in the partitioning or scheduling, the implementation has to be changed substantially. On the other hand signal processing implementations are highly individual and can hardly be automated by some tool process.

Additionally we were able to specify further requirements to the signal processing implementation, which substantially reduce the complexity of the control implementation and also simplify the implementation of the code generation. A simple example is the detection of a processing state if multiple processing chains are present: If an instance of a function reaches a state, it may notify this by setting a corresponding output value. As a consequence all other instances of the same function would subsequently set the same value too. As the control only needs to act on the first occurrence, some additional handling would be needed to suppress all subsequent notifications by the other function instances. Integrating this mechanism into the function itself, so that only one instance of the function signals the occurrence just once, simplifies the control implementation and therefore the code generation reasonably.

As the implementation of the code generation is in a early state, we successively raising the complexity of the processing chains, for which we generate code. Figure 4 shows a Simulink model, which is used to test the partitioning and scheduling as well as the code generation. It is a part of the 1 and 2 MBit receive chain of our WLAN 802.11b implementation and allows a simulation of the model. During the processing of this model, all blocks, which are only needed for the visualization of the simulation output, like scope and data conversion are omitted and only blocks relevant to the target system are preserved.

The next challenging step for our model based approach is the seamless switching between different processing modes, represented by different processing chains.

8. CONCLUSION

Throughout this paper we showed the need for a system level abstraction of parallel hardware properties, to achieve a decoupling of the application and the architecture design and to achieve an acceleration of the design process by code generation. A partitioning and scheduling solution as well as a code generation process was presented, based on a system level model. To avoid the reimplementation of functionality, which is already provided by available tools, new steps in the design process are developed as plug-ins using the Eclipse framework and by interfacing tools like Simulink as front-end and simulation environment for the application model.

The evolutionary extension of existing tools has the advantage of an immediate impact to the design process. Nevertheless the proposed extensions addressing fundamental steps within the design process, like the partitioning and scheduling, as well as the code generation for multithreaded or multi-core architectures. Hence these tool extensions can be reused with more suitable modeling languages.

The application of this approach to the modeling of WLAN 802.11b and its manual reference implementation already showed the positive impact of a code generation to the design process. Further investigations are needed to allow a code generation for more complex control tasks, like the switching between different processing chains and a detailed analysis of the savings in development time will follow up.

9. REFERENCES

[1] Eclipse Platfrom Technical overview, http://www.eclipse.org/whitepapers/eclipse-overview.pdf

[2] E. D. Willink, "The Waveform Description Language", in *Software Defined Radio*, pp. 365-397, Wiley & Sons, Sussex, England 2002

[3] Edward A. Lee, "Overview of the Ptolemy Project", Technical Memorandum No. UCB/ERL M03/25, University of California, Berkeley, CA, 94720, USA, July 2, 2003.

[4] Morgan Björkander and Cris Kobryn, "Architecting Systems with UML 2.0", IEEE Software, July/August 2003

[5] SysML Specification v. 0.85 Draft, http://www.sysml.org/artifacts/spec/SysML-v0.85R1-PDF-041011.zip

[6] F. Balarin and Y. Watanabe and H. Hsieh and L. Lavagno and C. Paserone and A. Sangiovanni-Vincentelli, "Metropolis: an integrated electronic system design environment", IEEE Computer, vol 36, no. 4, pp 45-52, apr 2003.

[7] M. Arnold and H. Corporaal, "Automatic Detection of Recurring Operation Patterns", CODES'99, Rome Italy, May 1999.

[8] H. Corporaal and H. Mulder, "MOVE: A framework for high-performance processor design, Supercomputing-91, pp. 692 – 701, Albuquerque, November 1991.

[9] Numerical C Extensions Group of X3J11 DPCE Subcommittee, "Data Parallel C Extensions", Technical Report, Version 1.6, X3J11/94-080, WG14/N395, December 31, 1994

[10] H.-M. Bluethgen and C. Grassmann and W. Raab and U. Ramacher, "A Programmable Baseband Platform for Software-Defined Radio", SDR'04 Technical Conference, Phoenix, November 2004

[11] M. Nunkesser, "Mapping Task Graphs to Digital Signal Processors", Technical Report TR 460, Department of Computer Science, ETH Zürich October 2004

[12] T. Grandpierre, C. Lavarenne and Y. Sorel, "Optimized Rapid Prototyping For Real Time Embedded Heterogeneous Multiprocessors", CODES'99 7th International Workshop on Hardware/Software Co-Design, Rome, May 1999

[13] W. Raab et al., "A Development System for Heterogeneous Multiprocessor Architectures", IEEE Workshop on Heterogeneous Reconfigurable Systems on Chip (SoC), Hamburg, 2002.

[14] S. Niemann, "Functional Modeling With UML", *Technical Report*, I-Logix, 2004

[15] J. Long, "Relationshiops between Common Graphical Representations in System Engineering", *Technical Report*, Vitech Corp., 2002