MODELING LANGUAGE FOR SOFTWARE DEFINED RADIO APPLICATIONS

Matthias Weßeling (BenQ Mobile, CTM PIC NGT, 46395 Bocholt, Germany, matthias.wesseling@siemens.com)

1. ABSTRACT

The mobile communication market is confronted with an increasing number of communication standards and a corresponding complexity for the mobile terminal applications. To cope with this complexity, the Software Defined Radio approach gains more and more attractiveness. The upcoming hardware platforms supporting a manifold of communication standards all have to compromise between the degree of supported flexibility and the required power consumption. These compromises cause a more complex programming interface, because parallelization potential, synchronization and latency time restrictions have to be evaluated in detail. The decoupling of the hardware platform from a specific application will also change the business models in the future, because software and hardware don't have to be developed from the same company anymore. But to support this development, a new interface has to be defined to separate the required know how about the hardware platforms from the know how about the application specification. This paper will introduce one proposal and show development results.

2. INTRODUCTION

One important architecture feature for currently proposed hardware platforms for Software Defined Radio is the parallelization of processing power in very different kinds [2,3,4]. But this parallelization also leads to additional programming restrictions, which requires a detailed consideration of the parallelization potential of the application. To reduce these complexities and to speed up the development process, we suggest the separation of the software development flow into at least two different steps, such that developers need the experience only in either the hardware/software or in the application area. In one development step, the application is specified in a hardware independent way, and in a separate step, the hardware specific implementation can be done without knowledge about the application.

This will allow to create new business models for programming Software Defined Radio platforms, because programming a mobile communication standard like UMTS, WLAN would not require any hardware know how anymore, and no complex programming with adaptation to any specific hardware details. The implementation step can be performed in a separate development step, and if the application description contains sufficient information, this step could be automated. To allow this automatism and an optimization in the implementation step, the application has to be specified in a way that the required processing is described in detail, but which still keeps the potential for hardware-specific optimizations. Within the SDR project at BenQ Mobile, we developed a new programming flow that supports this concept by defining a new modeling language and providing hardware specific adaptations compilation techniques.
Considering specific hardware architecture features, we will show how to describe algorithms to keep the flexibility for an efficient mapping to all different hardware platforms. The required structures for data flow and control description with the corresponding schedule information is presented. Additional to the language itself the implementation requires a few separate steps for the insertion of the hardware information and about the required programming interface. A tailored implementation of the described algorithms on the specific hardware platform should be possible.

With a complete description of WLAN 802.11a/b physical and MAC layer we can verify the concept of the programming flow and the programming language together with the development tools.

Additionally we will show, why and how the business model for programming mobile platforms will be different for software defined radio platforms compared to current platforms, and that a standardized hardware independent interface for these platforms will be basic for the market success.

3. CHANGE OF BUSINESS MODEL

The Software Defined Radio approach will not only change the separation of the hardware and software development, but also will change the future distribution of development between the different business areas. A semiconductor company, which will release a new hardware platform specific for a GSM or UMTS device, needed to create a lot of know how specific for this standards. This was implemented into the hardware, and the basic software will be implemented by this company, because hardware and standard know how is required for this development. The software development is restricted to this one standard, because the hardware does not support any other standard.

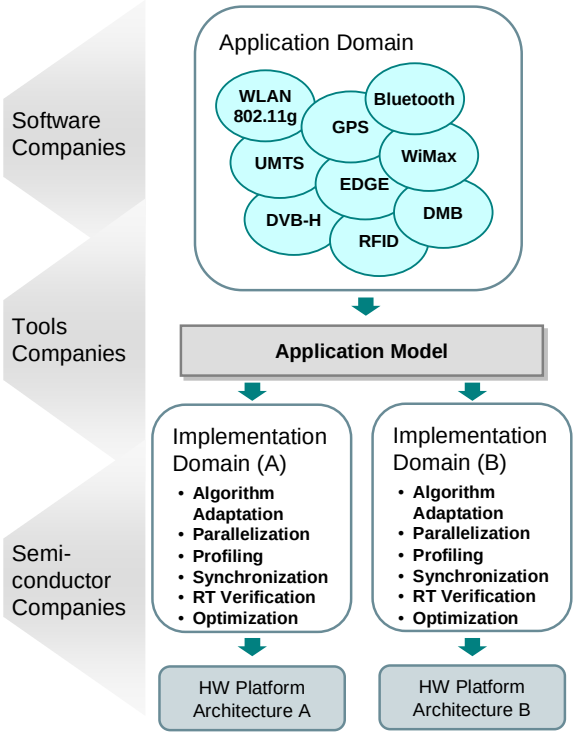

Figure 1: Application Modeling as Interface

As soon as the hardware platform now is independent of the supported standards, the know how of the standards is not required anymore for the hardware development. The same is true for the software development. So why should this work still be performed by a semiconductor. To utilize the potential of such a flexible hardware, a lot of different software implementations are required, or otherwise the flexibility is not seen by the customers. To cover this huge development area probably many different companies with many different know how areas are required. Figure 1 shows a possible separation between application modeling and implementation on a hardware platforms.

4. MODELING AND IMPLEMENTATION CONCEPT

Modeling of any systems always try to simplify a description and hide details of components to abstract the overall description. The essential question with good modeling is always the right separation between the information that is really required at the model level and the information which can be hidden under a component, because it has no impact on the description of one layer above the component. In our concept we try to separate as much as possible between the information that is required for the different development steps required to create an implementation on the specific hardware architecture. Looking at Figure 2 we distinguish between different description domains.

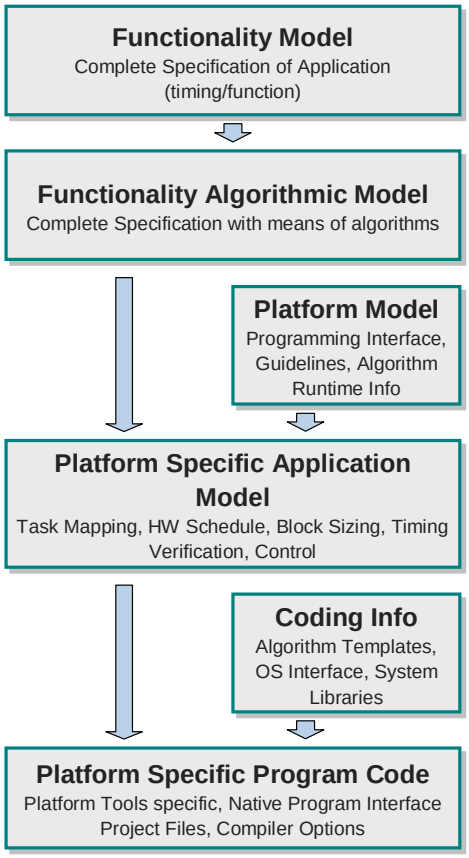

Figure 2: Modeling Domains

Functional Model of Application

The highest level is the 'Functionality Model'. This can be seen as the specification of an application covered by a model description. The information we see in this level of abstraction should be similar to the information of a specification document, but represented in a specific form and ready to be processed automatically. The abstraction

should be that high, that all implementations of this application could be derived from this functional model. The model should not contain any algorithmic description of the signal processing as long as it is not required. The filters are not designed in this document, only the filter performance is to specify. And whether a convolutional code is decoded with a viterbi decoder or any other one is not important for the application as long as the receive performance is achieved. Currently this level of description is not within the focus of our work and still is a broad area for future academic work.

Functional Algorithmic Model of Application

The next lower domain level is the 'Functionality Algorithmic Model', which is based on an algorithmic description that will meet the specification. Here we can define in detail, which filter to take, the over-sampling rate, the dynamic range of the ADC, the frequency tracking or channel estimator algorithms, etc. We should define anything that affects the functional operation, but have to be careful to keep a high potential for different implementations on different hardware platforms. Especially the control of the algorithms is a crucial point, because this differs in a huge range considering several hardware architectures (see also chapter 5). Also real-time conditions are part of the description, but since we are still hardware independent in this domain, they are descriptive only and are separate from the functional description.

Platform Architecture Model

For an efficient implementation we also need a model of the hardware architecture. This model does have to contain all the information required for the right implementation decisions. This is not necessarily the real hardware architecture, but more the interface to the programmer. Here we find a database of all supported algorithms, with exact timings for each possible implementation, the potential of parallelization, synchronization strategies and timings, or external interfaces together with timing restrictions.

Platform Specific Implementation Model

The information of the functional algorithmic model and the platform architecture model can now be merged into a specific implementation for this platform. All information that is required for efficient decisions have to be part of the one or the other source model. The 'Platform Specific Implementation Model' describes in detail, how to control the whole application, which part is mapped to what resources, what will run in parallel, and where do we need synchronization between parallel tasks, and what synchronization methods should be used. From this model we should see the exact run-time behaviour as long as it is not data dependent. All decisions about implementation compromises are already taken (see also Figure 6).

Implementation Program Code

This is the real program code and not a model as described in the previous domains. The program code is just another representation of the platform specific implementation model, but it is now in the input format required by the native compiler tools, which should create exactly the behavior that was described in the implementation model. For the most hardware platforms it is a C like code structure and depending on the architecture with vector or multi-task extensions. The required adaptations to the operating system, the supported system libraries and data types are required to create this program code. The generated code does not have to be restricted to the functionality itself, but can also contain all the project files, compiler options, dependency files or the makefiles.

5. A MODEL AND IMPLEMENTATION EXAMPLE

The most reasonable interface in the previous development flow we see in the functional algorithmic model of the application. The choice of algorithms can be of course already hardware dependent, because they are more or less optimal for specific architectures. Nevertheless, a change of algorithms on this high level is not a crucial point and can easily be done compared to the implementation changes on mobile platform architectures.
To see the requirements of a model language we will introduce a small application example and try to map this to two potential hardware architectures for mobile terminals. First we will have a look to potential architecture characteristics to be considered for mapping and scheduling strategies.

Architecture Characteristics

This architecture can be based on a large number of DSP units, all running with a relative low clock frequency. The synchronization of the DSP tasks, is done by software with no special hardware support. The strategy for the implementation in this case is to find as much parallel tasks as possible and trying to reduce the synchronization efforts, because latency time is a crucial point due to the low clock. With large vector execution units we have to find a high degree of data parallelism, which also affects the latency time. The level of potential vectorization can be very different for parts of the application. Another option are optimized accelerators for specific processing algorithms.

The flexibility is quite reduced and therefore the mapping of abstracted descriptions more difficult.

The Application Examples

We will introduce two small examples to show at least a few of the points that are important for the model specification and therefore the model language. The first one shows a few parallelization aspects and the second a control discussion. There are of course more points to consider, but this will give already an impression of the strategy we follow with our modeling.

Figure 3 and Figure 4 sketch a streaming application, without the outer control of the stream. It consists of only three different algorithms in a signal processing chain. Considering two different choices for a parallelization of this application the compromises to be taken are shown. In the first case in Figure 3 the data stream is sliced into several blocks and mapped to parallel execution units. For the filter operation we can only skip the synchronization of the internal states if we overlap the input data with the size of the filter length. The demodulation has no internal data and can therefore run independent in each task. But for the Viterbi operation we need synchronization, because the internal states have to be accessed in the right sequence.

In the second case (Figure 4) we parallelize the functionality, which changes the control a lot. The input stream of the filter now does not require any overlapped input, but now we have to synchronize between the functionality blocks. The granularity size of the synchronization (wait on 1 or n samples) should be dependent on the specific timings on the platform. Especially on platforms with accelerators, this structure will be applied.

We see a very different control in these implementations, which can't be part of the application model as long as we can not convert one control to another control description. Therefore the control should really be restricted to functional specifications. Also the block size of the components can be used for description, but only for descriptive reasons. It must be possible to convert these block sizes for tailoring them to hardware characteristics. One very important point is the scheduler, which is always strongly hardware dependent. This is a difficult issue, because we have to assume a scheduler for the description, but the description should also keep the potential to change the schedule as much as possible to cover a wide range of different architectures. A definition of a scheduler with as less assumptions as possible is an essential part of a model that can be mapped on many different architectures.

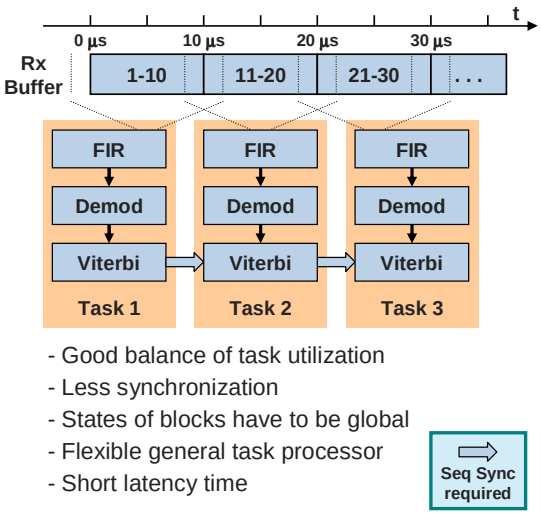

Figure 3: Parallelization of Data Stream

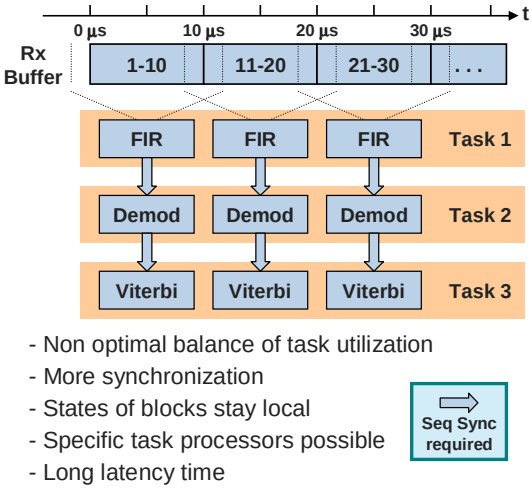

Figure 4: Parallelization of Functionality

We solve this issue with the following strategy. There are two different schedule domains. One is the Synchronous Data Flow (SDF) domain, which is also known from other simulation tools, like Cossap, SPW or MLDesigner. All blocks could run in parallel as long as they consider the data dependencies. There is no data transfer between blocks independent of this data flow allowed, for example by memory access. In this case the right access sequence would not be defined.

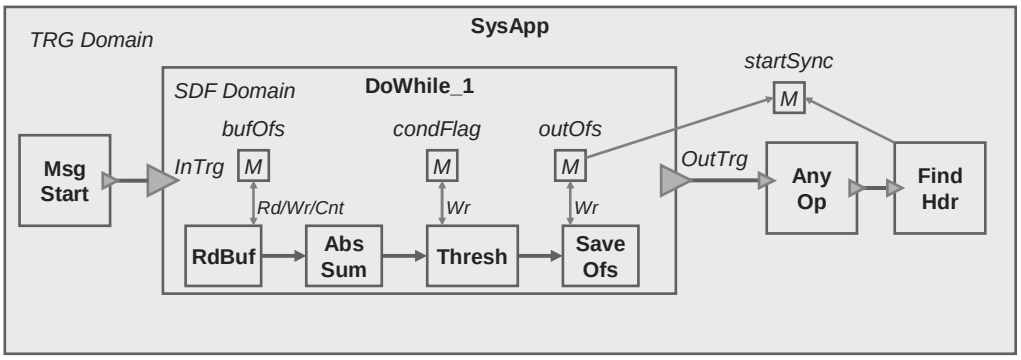

Figure 5: Control Description Example with a Do-While Loop

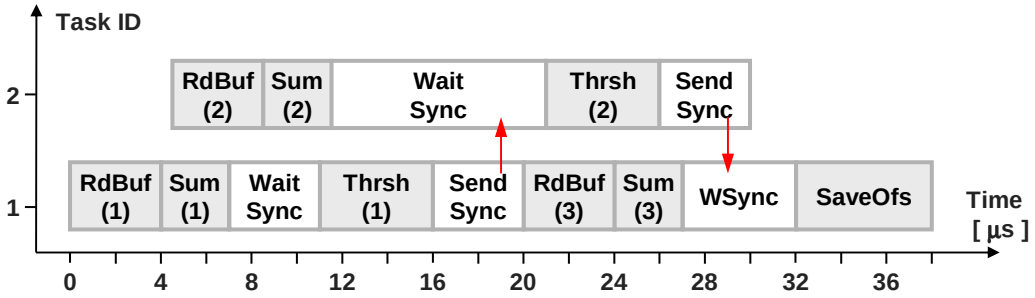

Figure 6: Mapping the Do-While Loop to 2 Tasks

The other schedule domain we define is the trigger (TRG) domain. In this case there is no data flow allowed between blocks, but just trigger exchanges. These triggers are just for describing the sequence of execution, but do not require an exchange of any particle. This way, there is one exact sequence how to execute the blocks, which allows arbitrary memory access. But additionally each dependency between blocks by memory accesses are visible in the application model and for the implementation step this can be analyzed and therefore the parallelization potential is still available. These domains can be mixed between the hierarchies, but not within one hierarchy level.

The next example is focused on a loop control. The description of Figure 5 takes the data of an input buffer, sums the absolute values and compares them with a threshold and stores the current buffer offset. This is repeated as long as the threshold is not exceeded. Once this loop is left, there can be processed the AnyOp block and then it starts with finding the header in the buffer. Trying to map this algorithm to a multi-task hardware platform we could select the following solution: The iteration steps within the DoWhile_1 block are mapped to two different units, that are processing the iterations in an alternative way. Unit 1 takes the odd loop iterations and unit 2 the even iterations.

The first problem is the RdBuf block, because it has an internal state for the buffer offset (bufOfs). The read and write access usually does not allow a parallel execution of succeeding loop steps. But in this case we know what will be written into this state and can use this information, to allow a parallel execution. Therefore this is important information and has to be part of the model. We describe this with a counter access type and additional access properties.

The usual do-while loop algorithm is based on the evaluation of a condition at the end of one loop iteration. This would avoid a parallel execution of several loop iterations. Therefore it is important to make it visible, where this condition (condFlag) is determined to evaluate this condition as early as possible. And when additionally the implementation is structured in a way that the changes performed within the iteration step can be undone, then we have a higher potential for parallelization options.

The DoWhile operation is specified just by convention of the block name and the memory item named condFlag.

In Figure 6 we see that the second loop step is started in task 2 before the first one has finished and the loop condition is evaluated. This can be done, because there is no change of global data. The only operation to be processed sequentially is the Thresh block. This is the information that has to be exchanged between the two tasks, and therefore requires a defined sequence. We framed this block with the WaitSync and the SendSync blocks. In this case we see that a further parallelization to 3 or more tasks would not increase the performance because the required synchronization would avoid an efficient parallel execution. But other synchronization mechanisms would be possible as long as we exactly see what has to be done for this application.

The saveOfs block should store the last buffer offset before the threshold was exceeded. Because this is a write only access to a memory, which is not used anywhere else in this loop we can skip this operation as long as it was not the last loop.

The AnyOp block does not have any dependency on the loop block and therefore we are free to execute this block before or in parallel to the loop iteration. In contrast to this the FindHdr block has a potential memory read/write conflict on the startSync memory. Therefore this block has to wait on the last activation of the saveOfs block within the loop.

The MsgStart block is required to define when the processing is to be started. The additional information is of course, how many times and in which time periods we will get a message to start the whole operation. If we receive such a message during the execution, the considerations regarding mapping to tasks will change significantly. The type of the description in the application is just a block with the name MsgStart and the message name. Whether this is a real message to be handled by a message handler, an interrupt, or any procedure call is of course dependent on the hardware architecture.

6. DIFFERENCE TO OTHER MODELING TOOLS

Modeling applications to abstract the implementation is not a new issue and therefore we have to argue why we do not use the solutions already available and why to develop our own solutions. The main argument here is that all the known simulation and modeling tools are quite efficient to describe a simulation of the algorithms of an application, but are all based on the specific schedulers. The descriptions are restricted to these fix defined schedules and a mapping to other schedules, especially for multi-task architectures is not possible without a very complex and detailed analysis of the description. Many schedule decisions are taken dynamically during the run time of the simulation, because the required information is not explicitly specified in the model. This is of advantage for a flexible and easy modeling, but avoids an efficient and tailored implementation.

Other important information is not covered by most of the tools. The real time requirements of the application itself, but also the real time behavior of the interfaces of the application model are needed to evaluate an efficient mapping. The streaming algorithms are covered quite sufficiently, but the control modeling is harder to abstract and still needs more consideration for automatic implementation design flows.

7. STATUS OF WORK

Within BenQ Mobile we have a complete model of a WLAN 802.11a/b transmitter and receiver for the digital part, and a basic tool chain, which can generate a complete and running code for a multi-task hardware architecture. These models are created with the simulation tools MLDesigner and Simulink, where specific guide lines where applied to create exactly the required information and make the models independent of the attached schedulers.

The next steps will focus on the timing considerations and the creation of optimized mapping and schedule algorithms.

8. REFERENCES

[1] R. Hossain, M. Weßeling and C. Leopold, “Virtual Radio Engine - A Programming Concept for Separation of Application Specifications and Hardware Architectures”, *Proceedings. 14th IST Mobile and Wireless Communications Summit*, Dresden, June 2005.

[2] 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

[3] C. Grassmann, M. Sauermann, H.-M. Bluethgen and U. Ramacher, “System Level Hardware Abstraction for Software Defined Radio”, SDR’04 Technical Conference, Phoenix, November 2004

[4] K. van Berkel et al., “Vector Processing as an Enabler for Software Defined Radio”, SDR’04 Technical Conference, Phoenix, November 2004

[5] S. Jinturkar et al., “Software Centric Approach to developing wireless applications”, SDR’04 Technical Conference, Phoenix, November 2004