

Using the Application Modeling and Mapping Methodology for System-Level Performance Analysis

René van den Berg, Walter Tibboel, Rob Wieringa, Martin Klompstra
NXP Semiconductors, Netherlands

rene.van.den.berg@nxp.com, walter.tibboel@nxp.com, rob.wieringa@nxp.com, martin.klompstra@nxp.com

Abstract

This article describes our experiences using the Application Modeling and Mapping methodology (AMM) based on commercial tooling from Synopsys. This methodology is valuable at the technical as well as at the organizational level for investigating the feasibility of new electronic products. Technically, the methodology reduces the risk by giving architects a clear understanding of the application and features in an early stage of the project. This is related to system performance, hardware and software allocation on available resources, software scheduling scenarios and architecture dimensions and decisions (what-if scenarios). On the organizational level, the methodology facilitates the early collaboration of system architects, software developers, and hardware designers based on an executable specification of the product. Within this article the AMM methodology is discussed and applied to a dual DAB reception application. For the different aspects as described above the benefits and disadvantage are shown and discussed.

Introduction

The continuous integration of more and more applications into a single device imposes huge challenges on the design of electronic products. Given the enormous cost of a SoC design project, the system architect has a very difficult task to define the right system architecture, which supports all the desired application use-cases in an optimal way:

- Over-designing the system leads to excessive cost and non-competitive products
- Under-design causes expensive chip re-spins until the performance requirements are met.

Hence there is a high risk associated with the dimensioning of the SoC architecture.

Unfortunately the performance analysis of such complex systems is not trivial, because at execution time multiple application use-cases utilize different resources of the SoC architecture. In addition there are multiple levels of resource sharing in the different components of the architecture:

- multiple tasks compete for programmable processing elements
- multiple components compete for access to the interconnect
- multiple ports of a memory controller compete for access to the shared memory

According to our experience, the usage of spread-sheets is no longer appropriate to analyze the performance of a dynamic application workload on a resource-shared architecture. To cope with the increasing complexity we have developed a simulation based performance analysis methodology, which allows the systematic exploration of application use-cases and SoC architectures. Our goal is to more accurately estimate the performance of the final product and this way significantly reduce the risk of over- or under-design.

In this article we present the results of a product analysis, where the methodology has been applied to a feasibility analysis in the area of Digital Audio Broadcasting (DAB). The goal of this analysis was to evaluate if an existing SoC architecture can support two concurrent DAB radio streams. In the following sections we will first introduce the performance analysis methodology as well as the application domain and the reference architecture. After that we discuss the results and conclusions of the feasibility analysis.

Application Modeling and Mapping Methodology

The performance analysis methodology is based on a new solution for system level performance analysis called Application Task Mapping (ATM), which Synopsys has added to their Platform Architect product line (formerly with CoWare). In essence ATM enables the rapid creation of an executable system model to collect information about performance metrics like throughput, latency, and resource utilization. This way the system architect can evaluate the feasibility of the SoC architecture and the application partitioning.

On top of the ATM capabilities we at NXP have developed a methodology called Application Modeling and Mapping (AMM), which applies the capabilities of the Synopsys Platform Architect environment to signal processing applications. This methodology comprises libraries for modeling of signal processing tasks and customized analysis views.

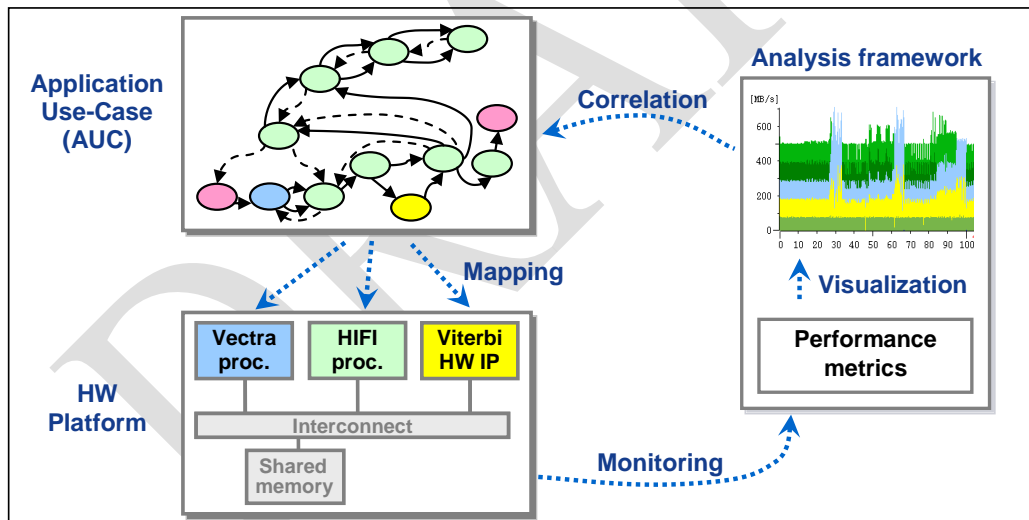


Figure 1: AMM methodology overview

As depicted in Figure 1, the AMM methodology covers the following aspects:

- Creation of executable SystemC task graphs representing the *Application Use-Cases*.
- *Mapping* of the *AUC models* on the *HW platform*.
- *Monitoring* of performance metrics of processing elements and communication fabrics.
- *Visualizing* performance metrics at the adequate level to identify bottlenecks.
- *Correlating* the obtained metrics with the processes/channels of the AUC model.
- All in an environment that allows short and fast iterations.

Application Use-Case Modeling

The *Application Use-Case* describes only the subset of the functionality that is needed for accurate modeling of processing time and traffic generation. It is basically an executable Data Flow Diagram (DFD) [\[link\]](#) inspired by Hatley & Phirbai design method [\[link\]](#). AUC models are based on the AMM library written in SystemC, which consists of the following primitives:

- A *Task* contains the functional code for the data flow, control flow, computation time and latency constraints. A task can be also hierarchical to manage the complexity of real-work application models.
- A *Data channel* models memory based communication between tasks. The channel contains a user defined buffer, varying from a single word to multiple video frames.
- A *Control channel* models event based synchronization between the tasks. The event messages can carry (small) parts of control information.

These primitives have been defined such that they are generic enough to support a compositional, top-down modeling approach: any existing implementation of data-flow communication and synchronization can be expressed with the AMM primitives. On the other hand the primitive set is sufficiently restricted to have clear semantics and to ease automation.

Hardware Platform and Virtual Processing Unit

The *HW platform* consists of a mix of approximately timed and cycle accurate SystemC models for memory hierarchy, interconnect and processing elements from the Synopsys SystemC model library. The key element here is the Virtual Processing Unit (VPU), which represents a generic configurable processing element for the SystemC task graph. Any instance of a VPU can act as a shared processor executing multiple application tasks or as a dedicated hardware block like e.g. a Viterbi accelerator.

The VPU models all performance related aspects related to the execution of the mapped tasks. This comprises the processing delay, the scheduling overhead as well as the generation of realistic bus transactions.

Mapping

Mapping the AUC model on a HW platform is realized by assigning the AUC tasks to the respective VPUs and assigning the channel to the respective memories. Channels between tasks assigned to different VPUs need to be mapped to a suitable SoC-level communication and synchronization mechanism. This is realized by so called *Drivers*, which convert the task-level communication of the AUC model into platform specific operations like e.g. an interrupt signal for control channels or a DMA transfer for data channels.

Separating the application model from the architecture enables the flexibility required for exploration, e.g. in case of DAB the flexibility to map one or two concurrent DAB streams onto one platform.

Monitoring and Viewing

The SystemC models in the HW platform (VPU, interconnect, memories) are instrumented with analysis monitors to measure the performance of mapped application use-cases. The monitors allow:

- Viewing *dynamic aspects* by recording detailed traces and aggregate statistical performance information over a configurable analysis interval.
- Recording *relevant metrics* like e.g. system-level end-to-end latencies and corresponding constraints.
- Recording at *right granularity / detail* to speed-up the analysis process.
- *Correlating hardware performance data* with the AUC processes and channels. This enables the system architect to relate hardware performance bottleneck like e.g. bus contention with the root cause in the application use-case.

The analysis information is recorded into a data-base, which can be post processed to quickly visualize the same data in different views, like e.g. plotting bus throughput over the simulation time-line. Examples of performance views are shown in the results section.

In the remainder of this article we describe the application of the AMM methodology to the performance analysis of a multi-mode DAB radio.

DAB Application Model

According to the AMM methodology, the first step is the creation of a SystemC task graph of the Application Use-Case (AUC). In this article we are looking at a Digital Audio Broadcasting (DAB) application, which is a digital radio technology for broadcasting radio stations. The performance analysis of a DAB system is not trivial:

- DAB has a number of country specific transmission modes (I, II, III and IV). Number, type, and duration of the received symbols differ per mode. For worldwide operation a receiver must support all four modes.
- A DAB stream consists of a multiplex of compressed audio and data sub-channels called a DAB ensemble.
- The user can select at runtime which DAB stream sub-channels are processed.

On the whole, the DAB application constitutes a very dynamic workload for the underlying SoC architecture. To investigate the performance of the complete system, the DAB AUC model needs to represent the performance aspect of all four DAB transmission modes in terms of processing and communication requirements. Finally the model must be compositional and extendable to enable performance analysis of future DAB features.

The AMM methodology is well suited to create a performance model of the target application. As depicted in Figure 2, the block diagram of the DAB application (upper left part) can be translated into the corresponding SystemC task graph. Note that the full complexity of the application is not shown, as most of the blocks are hierarchical and contain additional tasks.

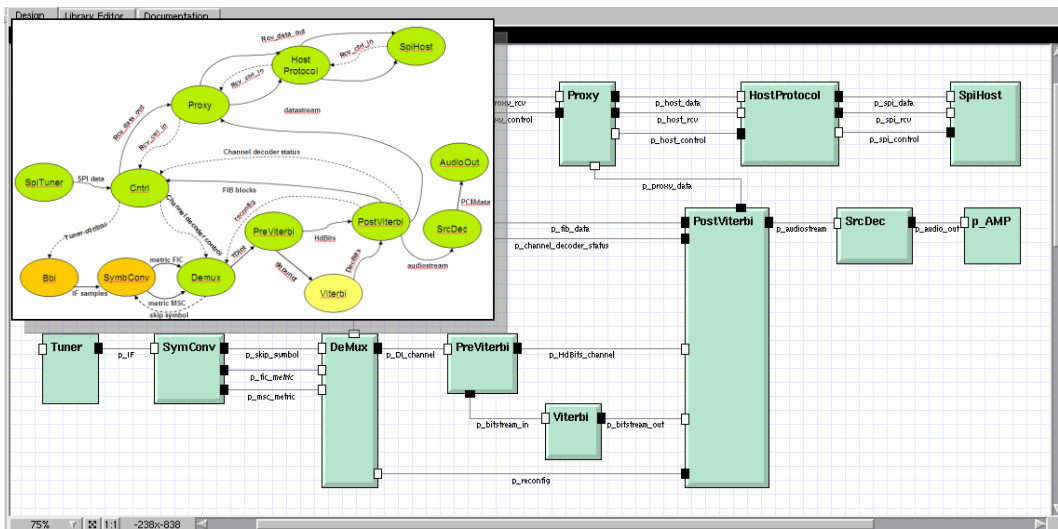


Figure 2: DAB algorithm, specification (upper left) and SystemC task graph

In our case the *Tuner* receives symbols and transfers the input samples to the receiver. The first task of the receiver is the symbol conversion (*SymConv*), which is composed of five sequential processes to perform down-sampling to demodulation. After that the de-interleaving and de-multiplexing of sub-channels is done in the *DeMux* block. The symbols are de-punctured in the *PreViterbi* before the *Viterbi* decodes the convolutionally encoded bitstream. The *PostViterbi* depacketizes the packet streams and routes them to *Control*, *Proxy* or *SourceDecoder*. The *SourceDecoder* decodes the audio samples and sends samples to the AMP terminator that represents the output speakers in the system.

The DAB AMM model is described in the SystemC programming language using the AMM methodology and libraries. The complete DAB AUC model has 25 tasks and 46 channels. The average effort required to create any of these processes is as low as just 200 lines of code. Typically, a process contains data and control dependencies as well as annotated information about execution time, data communication and synchronization. All aspects of the application model are dynamically specified by means of configuration properties. The Worst-Case-Execution-Times for the different tasks in the DAB AUC have been measured by the SW team.

The DAB AMM model is highly configurable and covers multiple application use-cases:

- All four modes of execution specified in the DAB standard are supported. Every process in the application model contains properties that can be configured with values according to the transmission mode.
- The synchronization aspects of the DAB application are accurately modeled.
- The model can process any DAB ensemble. The user can specify the ensemble of the DAB stream, i.e. the sub-channel composition of the DAB stream, and also which sub-channels have to be selected and processed by the DAB application.

The DAB AMM model can be executed standalone, that is, without being mapped onto a HW platform. Initially this is useful for debugging and verifying the functional correctness of the model. Once the application model behaves as expected, the standalone execution mode can be used to analyze performance aspects inherent to the application like e.g. the maximum application-level parallelism, data dependencies, and control loops. For this purpose the AMM library elements are instrumented with analysis monitors, which measure task activations and task-level communication. This information gives important insight for sub-sequent mapping of the application onto the SoC platform, e.g. the amount of data communication between the tasks can be used to dimension the required interconnect and memory hierarchy resources.

Reference hardware architecture

The SoC platform executing the DAB application is depicted in Figure 3. The chip includes a large number of programmable and dedicated IP blocks. However, most of the low-bandwidth peripheral elements like UART, I2C, timer, etc. are not relevant for the performance analysis simulations and can be omitted. This greatly shortens the modeling effort, because the platform model is composed of available VPU, interconnect, and memory models in the Synopsys library.

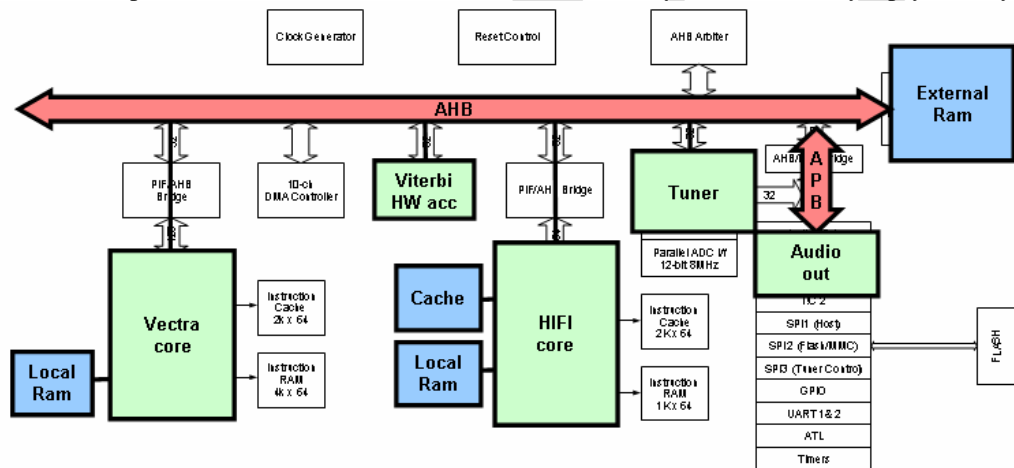


Figure 3: SoC Architecture with the performance relevant parts highlighted

The resulting platform model comprises only the elements highlighted in Figure 3:

- A VPU representing the Hifi processor core, including approximately timed local bus, memory, and bridge interface
- A VPU representing the Vectra processor, including approximately timed local memory and bridge interface
- A VPU representing Viterbi accelerator block
- A VPU representing the DMA controller
- Two VPUs representing two instances of the Tuner
- A cycle accurate model of the AHB interconnect and APB peripheral bus
- A approximately timed model of the global SDRAM memory controller
- The platform consists of 4 different clock domains

Since all SoC blocks are available in libraries, the effort for the creation of the SoC platform model boils down to the instantiation, connection, and configuration of the components in the Synopsys Platform Architect GUI.

Special care needs to be taken for the specification of the timing related configuration parameters of the approximately timed local bus and memory components. The timing parameters can either be configured from data-sheets, or can be measured from existing designs (RTL simulation or silicon).

DAB Application Performance Analysis

In this section we describe in detail the findings of the performance analysis project, where the AMM methodology has been applied to the design of DAB design.

On our case the silicon for a single DAB product already exists. The driving question for this performance analysis project is, if the same silicon can also support the processing of a dual-DAB application use-case. The advantage of having a reference design available is that we are able to measure the accuracy of the results obtained from the AMM performance analysis methodology.

Project Objectives and Scope

Before starting to create of the system performance model, the objectives and the scope of the performance analysis project should be clearly specified. This comprises the set of relevant models together with the necessary configurability and accuracy as well as the required analysis views. A well defined scope minimizes the modeling effort, since the modeling and analysis instrumentation can focus on the functionality that contributes to the project outcome. The scope should reflect the performance worries of the architect and the performance risks of the project leader of the product under analysis.

In case of the DAB application, the dual DAB stream is the most demanding application use-case for the underlying SoC platform. Especially the high-level interleaving schedule of two DAB streams is expected to be critical.

Within dual DAB setting, multiple configuration options exist. One important option is the ability to do audio source decoding on both streams (DAB 2) or on one stream only (DAB 1.0). The main objective of the performance analysis project is to validate the feasibility of these configurations.

In general a performance analysis project is an iterative process: new analysis results are discussed with the architect, which typically leads to new configurations to be analyzed.

Application Mapping and Test Setup

In this step, the DAB application model and the platform model are brought together to create a system performance model of the product. The application tasks are mapped onto the Virtual Processing Units (VPUs) of the platform as follows: the *SymConv* task is mapped on the Vectra VPU, the *Viterbi* task is mapped on the Viterbi accelerator VPU, and all other tasks are mapped on the Hifi VPU.

The results presented in the next section are taken from a use-case scenario, where two DAB streams with different transmission formats are simulated for 300 ms.

The first DAB stream is configured to run in mode 1 with these data rates:

- A DAB Audio service at 384 kbps (code rate 1/3)
- Two streaming data service at 308 kbps (205 + 103 kbps; code-rate 0.8)

The second DAB stream is configured to run in mode 3 with these data rates:

- Two streaming data service at 359 kbps (205 + 154 kbps; code-rate 0.8)

The total data rate is $384 + 308 + 359 = 384$ (audio) + 667 (data) = 1051 kbps. This combination has worst-case behavior.

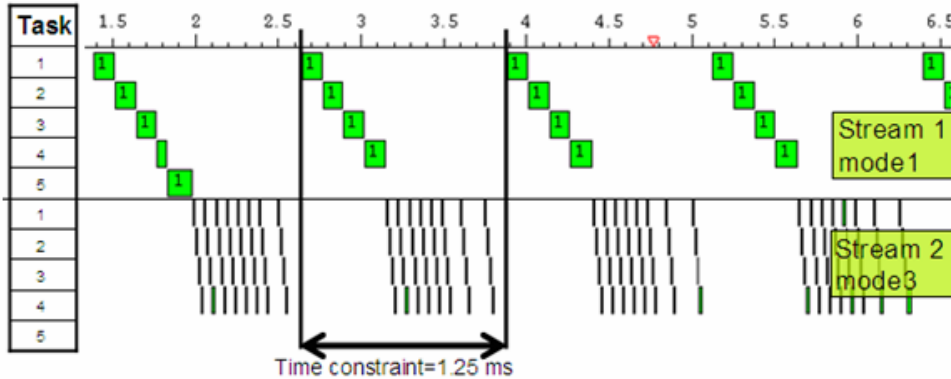


Figure 4: Tracing of "Dual-DAB" Application Use-Case with the concurrent processing of 2 DAB streams

The task trace in Figure 4 [Error! Reference source not found.](#) shows the concurrent execution of both DAB streams on the Vectra. A DAB stream of mode3 has eight times more symbols than a mode1 DAB stream. At the same time the size of a mode3 symbol is eight times smaller than a mode1 symbol, which implies that the overall number of transmitted bytes is the same for both streams.

Results

In the following we present the results from 2 phases in the performance analysis project. The first attempt to map the dual-DAB application onto the SoC platform did not meet the performance requirements. After an investigation of the initial results we performed several optimizations and finally managed to achieve our requirements.

Initial Mapping

The AMM simulation results showed that the dual-DAB application use-case did not execute successfully. The original assumption of the system architect was that the AHB backbone bus would be the primary bottleneck. The actual AHB bus utilization as depicted in [Figure 5](#) [Figure 5](#) reveals that the average bus load is only 62%. The different colors encode the different initiators, where the main contributor with ca. 44% is the Hifi processor.

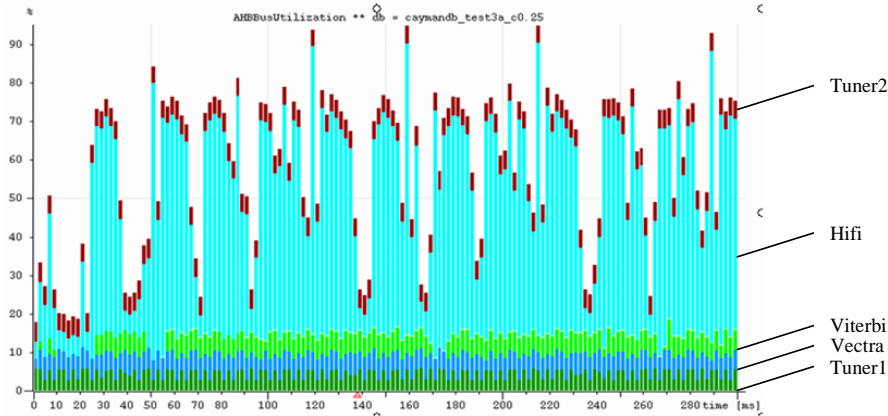


Figure 5: AHB Bus utilization

The reason for the performance limitation was found in the analysis of the Vectra processor. The upper part of [Figure 6](#) shows the utilization of the Vectra processor over time, where the different colors indicate the different tasks running on the VPU. The load varies significantly and the average utilization is about 80% percent.

More critical than the average utilization is the fact, that the application-level latency constraints are violated. This is shown in the lower part of [Figure 6](#): on a regular basis the processing latency is higher than 100% for more than 2 consecutive intervals. Every time this happens the maximum allowed end-to-end processing delay of the second DAB stream is exceeded. A closer analysis of the task execution delay reveals that tasks for the second DAB stream on the Vectra need significantly longer to run until completion than the same tasks for the first DAB stream. This mismatch originates from the long transaction latencies on the bus, because the symbols for the second stream do not fit into the local memory of Vectra. As a result, the active tasks of the second DAB stream spend most of their time waiting for data transfers to external memory.

Formatted: Font: Not Bold

Formatted: Font: Not Bold

Formatted: Font: Not Bold

Formatted: Font: Not Bold

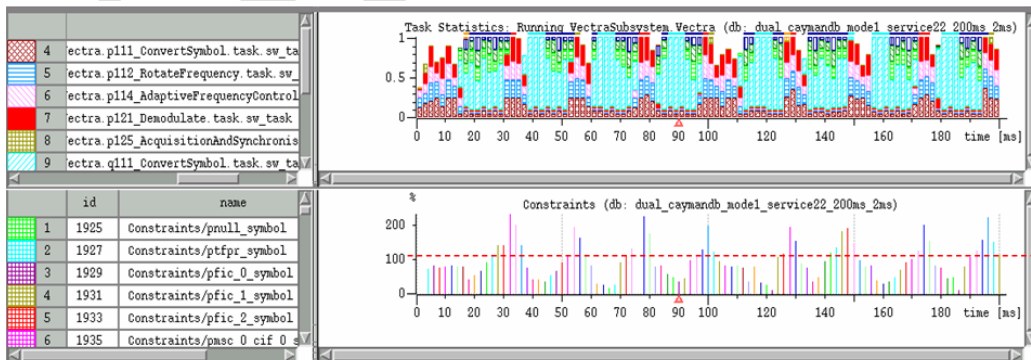


Figure 6: Vectra processor utilization and constraint violations from initial mapping

In summary, the initial attempt to map the dual-DAB application use-case of the available silicon platform did not meet the application level performance requirements. However the results from

the AMM simulation clearly indicated which parts of the application needed to be optimized in order to meet the requirements.

Final Mapping

Based on the results from the initial mapping we optimized the processing of the Vectra tasks and modified the mapping of the DAB data channels into the SoC memory hierarchy. After that we were able to store the symbols for both streams in the Vectra local memory.

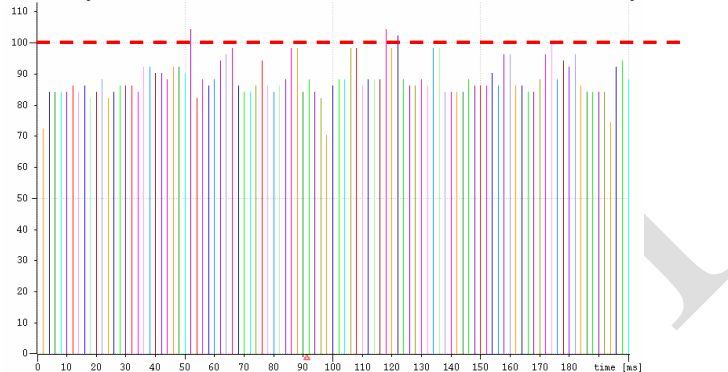


Figure 7: Final latency of Vectra Processing in percent of available timing budget.

This mapping meets the end-to-end performance requirements although occasionally the processing of individual symbols takes slightly longer than 100% of the available timing budget (see Figure 7). This is because longer latencies are immediately compensated by shorter ones, so the processing of an entire DAB frame is still within range.

In summary the parallel processing of two DAB streams running in different modes is now feasible.

Summary

This section summarizes our findings in deploying the AMM methodology in a real project. We discuss the accuracy and organizational aspects and after provide some concluding remarks.

Accuracy

The accuracy of the obtained simulation result is a key concern, since design decisions are based on these results. In our case the implementation of the previous generation DAB product are available as a reference, so we are able to measure and compare the actual performance numbers against the simulation results. Table 1 shows the average access latency measured in cycles per word for different processors and memories.

Table 1: Silicon test results versus simulation results

Processor	Memory	Silicon (cycles/word)	AMM (cycles/word)	Deviation (%)
Vectra	Vectra-local	2.6	2.4	-7.7
Vectra	Global	25.1	24.0	-4.4
Hifi	Global	20.5	20.0	-2.4

The deviation between the performance figures measured on the real silicon and those measured on the platform models is in all cases below 8%. This level of accuracy is by far good enough to assess the feasibility of application use-cases on a targeted SoC architecture.

Organizational awareness

At the outset of an AMM based performance analysis project all the information about product requirements, the application, and the HW architecture needs to be collected. As a result system architects, hardware designers and software developers are working together early on based on an executable and unambiguous specification of the system. This way the AMM performance analysis methodology reduces the risk for misinterpretation of textual specifications and facilitates the communication between different groups.

Conclusions

We found that the AMM methodology is very useful for early architectural exploration of complex designs. The accuracy between silicon results and AMM is less than 8%, which enables early design decision based on reliable numbers. The methodology gives easy access to the application and knowledge building between the different disciplines (system, hardware, software). High visibility can be reached based on the performance analysis monitors and correlation of application use-cases and HW resources.

In summary the AMM methodology together with the Platform Architect tool from Synopsys has proven itself to enable reliable performance analysis. Especially the quantitative investigation of processing delays and the detection of application-level constraint violations for highly dynamic use-cases is not possible with traditional methods like static spread-sheet analysis. Based on the simulation results we were able to optimize the mapping of the DAB application onto an existing SoC platform up to the point where the performance requirements are met even for the most demanding use-cases.