System level modelling with open source tools

Mikkel Koefoed Jakobsen[†] (mkoe@imm.dtu.dk) Jan Madsen[†] (jan@imm.dtu.dk)

Seyed Hosein Attarzadeh Niaki[‡] (shan2@kth.se) Ingo Sander[‡] (ingo@kth.se)

Jan Hansen[*] (jan@real-ear.com)

[†] Technical University of Denmark
Copenhagen, Denmark

[‡] KTH Royal Institute of Technology
Stockholm, Sweden

[*] Real Ear auditdata
Your Partner in Audiology Solutions

Abstract

In this paper, we present a system level design methodology which allows designers to model and analyze their systems from the early stages of the design process until final implementation. The design methodology targets heterogeneous embedded systems and is based on a formal modeling framework, called ForSyDe. ForSyDe is available under the open Source approach, which allows small and medium enterprises (SME) to get easy access to advanced modeling capabilities and tools. We give an introduction to the design methodology through the system level modeling of a simple industrial use case, and we outline the basics of the underlying ForSyDe model.

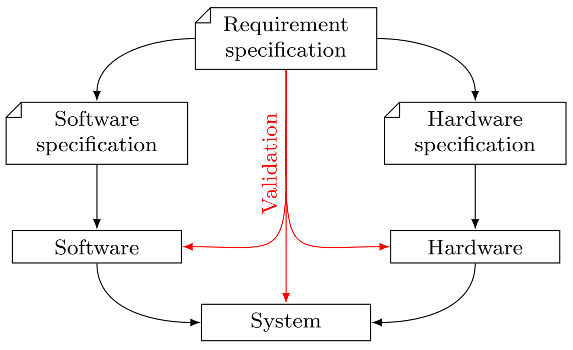

Figure 1: Current design methodology.

1 Introduction

Industry is facing a crisis in the design of complex hardware/software systems. Due to the increasing complexity, the gap between the generation of a product idea and the realization of a working system is expanding rapidly. To manage complexity and to shorten design cycles, industry is forced to look at system level languages towards specification and design. Such languages allows the designer to capture the system functionality from the very early stages in the design process and to use this system model as a basis for evaluating design decisions and for a stepwise refinement of the system specification into a final implementation.

Figure 1 shows the classical design flow. The initial requirement specification, which captures both the functional and non-functional properties of the system, is partitioned into a software and a hardware specification. This partition is usually based on the experience of the designers and on availability of existing hardware platforms. Although this may be a good starting point, the lack of being able to explore different design alternatives in a systematic way, seriously impact the quality and competitiveness of the resulting solution.

In this paper, we present a system level modelling approach aimed at capturing the early stages of the design, allowing the designer to explore tradeoffs and support design decisions. The proposed modeling approach is captured in a system design framework (SFF: System Functionality Framework), which has been developed as part of the SYSMODEL[1] project. The aim of this project has been to support the competitiveness of small

[1]Funded by ARTEMIS JU

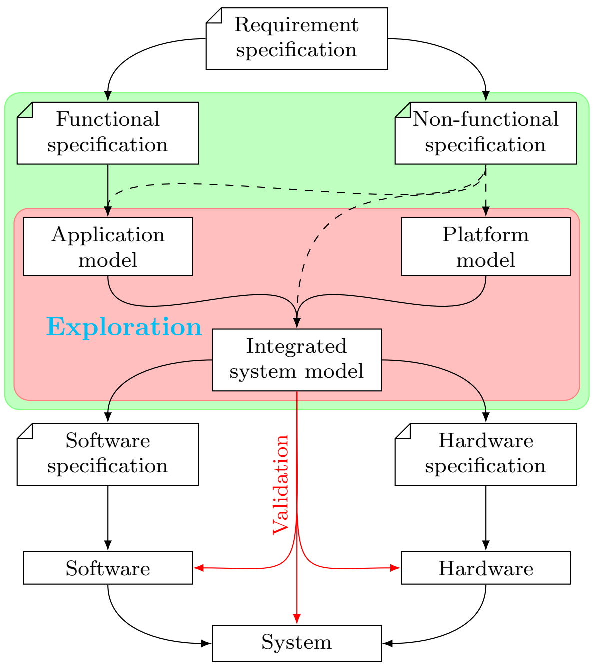

Figure 2: Design methodology using system modelling. Green area symbolises the modelling part of the design process. The red area is the design exploration part.

and medium-sized enterprises (SMEs). The general availability of the design framework is facilitated by the Open Source approach, where all tools are made available free of charge. Figure 2 shows how the proposed modeling approach extends the classical design flow with system level modelling. The initial requirement specification is partitioned into a functional specification and a non-functional specification. The functional specification is first translated into a suitable model of the application. Relevant non-functional properties, such as latency and power consumption, are used to guide this translation. Likewise, the non-functional properties are used to guide the selection of an appropriate execution platform, expressed as a platform model. Finally, the application model is mapped onto the platform model in order to form a model of the integrated system. As there are many ways to achieve this mapping, there is a need for being able to perform an exploration of the design space.

The proposed SFF framework is based on ForSyDe (Formal System Design) [4], a formal design methodology which allows several models of computation to be integrated in a single heterogeneous model, in order to capture and model different types of components, such as analog, digital and software components, and to describe a system at different stages in the design process. A formal modelling approach with clear semantics, makes it possible to formally reason about properties of the design, such as risks, price, power, and timing, already in the early stages of the design process.

We illustrate the design methodology outlined in Figure 2 through a system level design of a simple use case, a hearing aid calibration device. As the calibration device is a medical device, it has to apply to medical safety regulations [1, 3], which is always a challenge. One of the major advantages of the device as compared to competitors, is its small size and ease of use, which adds to the challenges. in order to work correctly according to safety regulations and medical specifications for hearing aid calibration devices, strict timing requirements are given. Now one of the key challenges in the early stage of the design process, where the complete system is being designed, is to ensure that a given application design when executed on the selected platform will always meet these timing requirements.

In the following, we first describe the use case and how the initial requirement specification may be transformed into an application model, and how this model may be bound to a platform model, in order to form an integrated system model. We explain how the models may be validated through simulation. After the use case, we turn to the modelling framework and outline the basics of the ForSyDe model. Finally, we give a summary and some concluding remarks.

2 Industry case

Throughout the paper, we will use a hearing aid calibration device as a use case for our proposed design methodology. The use case is provided by the Danish company Auditdata. Figure 3 shows the calibration device in context, i.e., the physical setup, where the calibration device controls sound generation and samples the sound in the ear through two microphones, one in front of the hearing aid device and one inside the ear right behind the hearing aid.

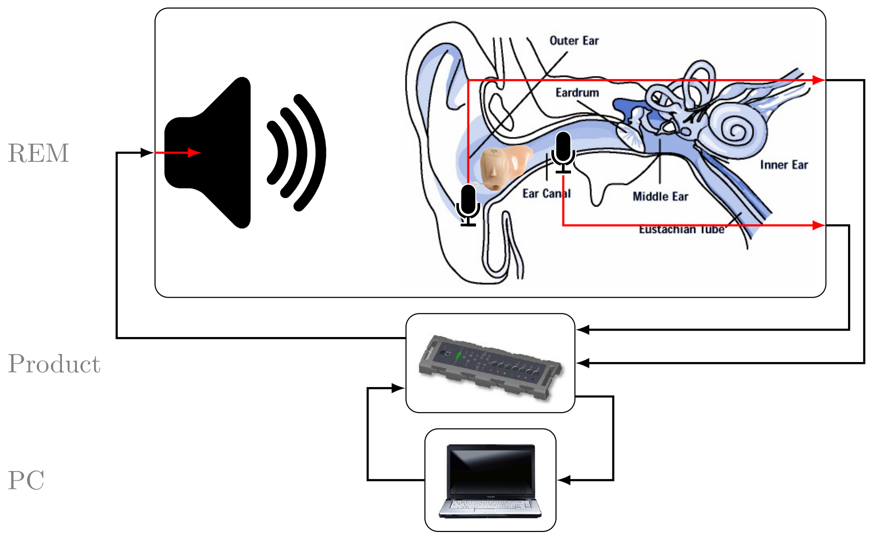

Figure 3: The problem that needs to be solved. A patient (the ear) needs to have a new hearing aid adjusted. The doctor places two microphones on the patient, one inside the ear canal and one just outside. In the room there is a loudspeaker. This setup is called a Real Ear Measurement (REM).

This is called the Real Ear Measurement (REM). The sampled sound signals are processed in the calibration device and send to the doctors PC via an USB interface for display. Figure 3 only shows the setup for one ear, however, the calibration device is able to handle both ears at the same time, with a total of sampling four microphones (using the same speaker for both ears).

2.1 Functional specification

The desired functionality of the product is to produce sound streams that are played in a loud speaker and record by up to four sound streams simultaneously that are then displayed on a computer screen as histograms in real time.

2.2 Non-functional specification

As the calibration process involves the patient to be able to relate visual and audio input, the calibration device has certain timing requirements. Furthermore, medical safety regulations require the signal processing to be done under real-time requirements and to deliver a certain accuracy. These timing and accuracy requirements, together with a wish to produce a low cost and low power device that are connected to the PC through a USB interface, challenges the design of the calibration device.

3 Application model

The functional part of the requirement specification can be expressed as an application model (block Application model in Figure 2). The benefits of such a model are that it can be used to validate the application behaviour, for instance by simulation of the model. Depending on the formalism used to describe the model, formal verification of system properties may also be a possibility.

3.1 ForSyDe model

An application model of the use case is shown in Figure 4. This particular model models the behaviour of producing one continuous sound stream for a speaker while continuously sampling two sound streams. Each of these sampled sound streams are transformed into histograms by a fast Fourier transformation (FFT)[2]. This represents one scenario/configuration of the product.

[2]A fast Fourier transformation is an efficient algorithm to compute the discrete Fourier transformation.

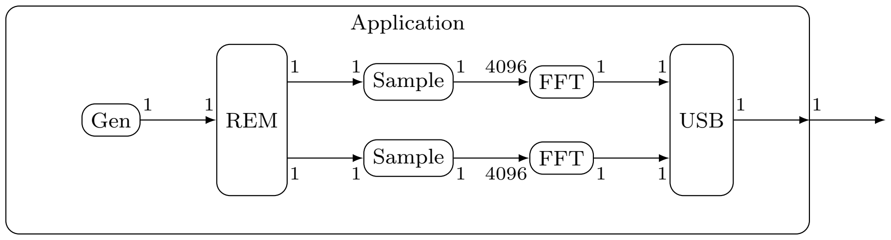

Figure 4: Simplified behavioural model of the product. This is a setup where two channels are actively being used.

In order to simulate this model, a standard FFT implementation can be used for each of the FFT blocks. This will produce the same functional behaviour as a version which have been optimised with respect to a given platform and non functional requirements.

We use a Synchronous Data Flow (SDF) [5] as the underlining Models of Computation (MoC). Synchronous Data Flow models the dataflow between processes and is hence, well suited for modelling streaming applications. Each process consumes a static number of tokens on each input and produces a static number of tokens on each output. A process will execute when sufficient tokens are ready on all inputs. The Models of Computation is without any notion of time. SDF can relatively easy be analyzed for required buffer sizes between processes. In the use case in Figure 4 the calculation of buffer sizes is not complex (maximum 4096 tokens before each FFT), but in more complex situations, e.g., where interaction would happen between the two streams of sampling followed by FFT, the analysis could quickly become much more complex.

3.2 Simulation

The application model can be simulated with very simple definitions of each process. This particular application model is simple enough that the behaviour is obvious. However, in more complex applications, only the behaviour of sub parts may be obvious while the global behaviour might not. Simulation can then be used to validate the behaviour.

An example of a simulation for this use case is to feed the application model with sine waves in the “Gen” process and verify that one frequency is dominating the output. Due to the discrete calculation of the Fourier transformation, the tone might not always be transformed into only one frequency, but will cover a small band. A result from a simulation of the application model is shown in Figure 5.

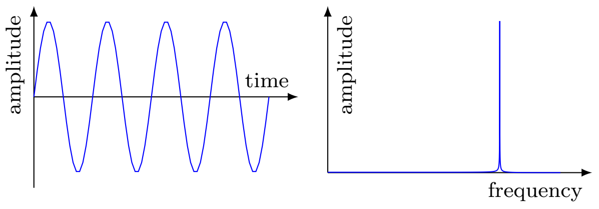

Figure 5: Example of simulation input and result.

The “REM” process can be used to model different changes to the sound as it travel through the air and ear channel.

3.3 Verification

In broad terms the application model can be used to estimate non-functional properties, verify that non-functional requirements are met, and explore behaviour to achieve certain non-functional properties. Such properties could be buffersizes needed for communication between processes or a static schedule of the processes. The verification techniques used to determine the properties depends on the MoCs used to model the application. It is possible to combine multiple MoCs in an application model, but some verification techniques will then only apply to certain parts of the application model.

Since the application model of the use case is modelled using only one Model of Computation (MoC), specifically the untimed MoC Synchronous

Data Flow (SDF), it is possible to perform verification on the entire structure of the model. One such verification is a buffer analysis, i.e. how much memory is (at most) required to contain all the data which is streamed through the device while operating. As an example, the difference between a 16MB and a 64MB memory solution is approximately a factor 4 in energy consumption (respectively 0.125W and 0.5W). The power used by the memory can therefore be a significant amount of the total power budget of the 2.5W provided by a USB port at maximum.

4 Platform model

The SFF framework can be used to represent details of all components and interconnections of the platform, i.e. a detailed representation of the hardware circuits. However, at the early stages of the design process, we are more interested in having a high level system model, which captures how the application interacts with the platform. In other words, the platform model provides the execution details, such as schedule and duration of the various application processes. In this context, the platform model defines the execution of the application model. This may be done by connecting the application model to the platform through control signals. Control signals from the platform model to the application model releases a process in the application model, and the opposite returns the control to the platform.

A platform model may start as a relative high level model with very few details. During the design process, it may be gradually refined with more and more details. For a platform containing a single processor core, may simply model the sequence of the application processes. This can be done by sequencing the control output from the application model to the next control input to the application model. A more complex platform model which includes some model of an operating system, may make more elaborate decisions on which application process to release, effectively modeling the behavior of a dynamic execution sequence, such as a fixed priority based real-time operation system. The execution semantics of SDF, ensures that the application processes will not execute until both data and control input are ready. Since the application model is without any notion of time and the timing of the execution of application processes are platform dependent, the platform is annotated with execution time of each application process.

In our use case, the platform is given and based on a single processor core. This core executes all digital signal processing and controls the A/D and D/A converters. The interface to the PC is done through a USB chip, which only has a single packet buffer for receiving data. The platform may also have to capture the behavior of an operating system which in our use case is described as round robin cooperative multitasking. In our case it simplifies into a static series of function calls to each application process.

As the application is presented as four streams (the pairs of sample and FFT processes), it may benefit from a platform supporting multiple cores. Hence, it would be interesting to explore alternative platforms, such as a platform with two processor cores or even four processor cores, each servicing a single stream. Although this may seems obvious, a challenge is that the operating system or the USB process system may be more complex, since synchronisation between the processor cores has to take place when collecting data for transport to the PC. Only a careful exploration of the design space will reveal the best tradeoff.

5 Integrated system model

The integrated system model is the combination of the application model and the platform model. The integration of the application model is performed by adding extra control dependencies to each process, such that the platform activates the process at the appropriate point in time. The estimated execution time of each application process is modelled through these control dependencies. These control dependencies are shown in Figure 6 as red arrows.

The platform controls the execution of each process in the application model by sending a release for execution through the input control dependencies. The output control dependencies to an application returns the control to the platform.

An important property of this modeling approach is that the application model with annotated control input/output dependencies, is independent of the actual platform. This means that

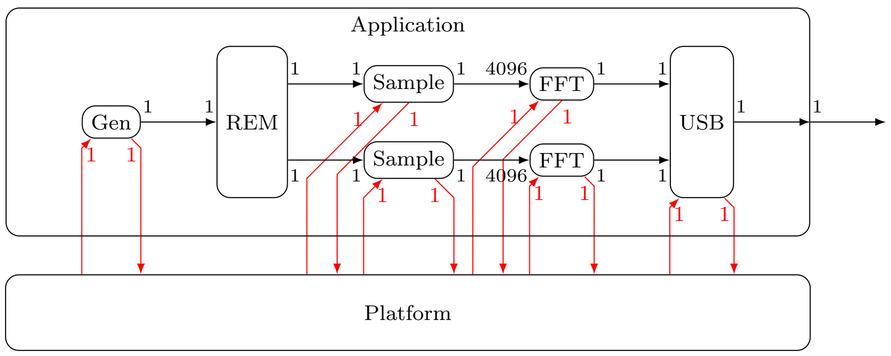

Figure 6: Integrated system model, the combination of the application model and the platform model.

the application model and the platform model are kept separate in the integrated system model, effectively applying a separation of concerns.

Furthermore, this allows us to perform what-if analysis of possible design choices. We may explore possible changes in the mapping, the platform or even in the application itself. Such explorations could be based on simulation or on an analytical approach which would allow for fast automated design space exploration. Hence, the aim of the exploration is to find the best solution which fulfills all non functional requirements. However, It is also possible to evaluate other relevant parameters of the design. Parameters which are not strictly being expressed as requirements, but are secondary optimization goals, such as the sensitivity of the proposed design. I.e. evaluating the amount of slack in the design which can be used to adjust the final solution in order to compensate for inaccurate estimates made in the early stages. This may lead to better solutions than those obtained from applying very conservative rules, which often leads to over designed systems.

5.1 Simulation

Simulating the integrated system model, can provide important insight into the design. We may be able to decide if the platform supports the application with the given requirements on execution time, power consumption, etc. Another important aspect of simulating a model of the system, is that we can easily get access to information, such as signals, components, variables and software blocks, which may be inaccessible in the final implementation. Further, we may use this information to easily infer start and end times of application processes as well as the preemption of these.

6 The ForSyDe framework

ForSyDe (Formal System Design) [4, 7] is a formal design methodology that targets heterogeneous embedded systems. ForSyDe uses the theory of models of computation (MoCs) [6] as its underlying formal foundation, which gives access to powerful analysis techniques during system design. ForSyDe designers do not need to have expertise in the underlying formal foundation, but will have access to the ForSyDe library, which encapsulates the mathematical base of ForSyDe. ForSyDe provides libraries for several MoCs, which allow to develop executable system models from which an analyzable mathematical model can be extracted. This analyzable model can then serve as a base for different tools in the following phases of the design flow, such as design space exploration and synthesis.

6.1 System Model

Figure 7 illustrates the ForSyDe system model. A system is modeled as concurrent process network, where processes belonging to the same MoC communicate via signals. ForSyDe system models do not have a global state, only local states are allowed. Rules for individual processes and mechanisms for concurrency and composition are defined within each MoC. At present ForSyDe pro-

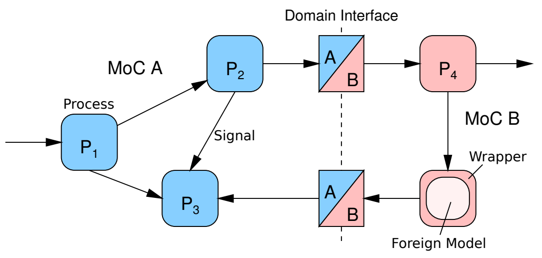

Figure 7: A ForSyDe System Model

vides libraries for four different MoCs, which allow to model heterogeneous embedded systems containing not only software but also digital and analog hardware at an abstract level: synchronous dataflow (SDF) MoC, synchronous MoC, discrete-time MoC, and continuous-time MoC. Processes belonging to different MoCs communicate via domain interfaces, which define how signals from one MoC are interpreted in another MoC. A typical example for a domain interface is an abstract analog-to-digital converter, which connects the continuous-time MoC to the synchronous or the discrete-event MoC.

Every novel design methodology has to cope with the existence of existing models or legacy code. Since these foreign models are not based on the ForSyDe formalism they cannot be treated as ForSyDe processes. However, using ForSyDe wrappers foreign models can be integrated into an existing ForSyDe model and co-simulated with the 'pure' ForSyDe processes (Section 6.4).

6.2 Process Constructors

A key concept in ForSyDe is the concept of process constructors. Process constructors lead to well-structured system models, which can then easily be converted to mathematical descriptions for which powerful analysis and synthesis methods exist.

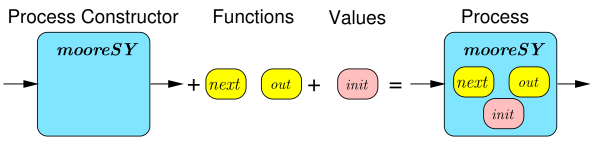

Figure 8: Process Constructor *mooreSY*

Each process is created by a process constructor. The process constructor defines the MoC, its interface to the environment, and a number of arguments that have to be supplied to the process constructor. Figure 8 illustrates this concept by means of the process constructor *mooreSY*, which is used to model a finite state machine and belongs to the synchronous MoC. *mooreSY* takes two functions and a value as arguments. The function *next* returns the next state of the state machine based on the present state and the current input values, the function *out* returns the output value based on the present statue, and the value *init* gives the initial state.

The ForSyDe methodology obliges the designer to create processes using process constructors. This gives several important benefits: (1) A system model is well-structured and well-defined, because each process constructor has a mathematical formulation, (2) process constructors separate communication (process constructor) from computation (function), (3) process constructors can have an implementation pattern (the process *mooreSY* can be efficiently implemented in software or hardware using a well-known design pattern).

6.3 Implementation of the ForSyDe Library

ForSyDe has originally been implemented in the functional language Haskell. Haskell fits perfectly with the formal foundation of ForSyDe, since it enforces side-effect free processes, provides lazy evaluation, and allows to express the process constructors with higher-order functions. The Haskell implementation supports four MoCs, and provides a hardware synthesis back-end, which allows to generate synthesizable VHDL from a synchronous MoC ForSyDe model [4].

After demonstrating the potential of ForSyDe with Haskell, a SystemC version of ForSyDe has been developed within the European Artemis project SYSMODEL [8] to increase industrial usability. SystemC is an industrial standard and widely used in industry for system modeling. However, SystemC lacks a clear formal semantics, which makes it very difficult to use it in its plain form for other purposes than modeling and simulation. Inside the SYSMODEL project we have created SystemC libraries based on ForSyDe for four MoCs, which ensures that ForSyDe SystemC models have a formal base and that abstract analyzable models, such as SDF-graphs, can be easily extracted from an executable ForSyDe SystemC model. Although compared to Haskell SystemC suffers from some drawbacks, in particular SystemC does not enforce side-effect-free processes, there is in addition to industrial acceptance another very important advantage: SystemC is a C++ class library and all functions are written in C/C++, which allows to directly implement the function arguments to the process constructors as C-code on the target processors.

The ForSyDe SystemC libraries implement process constructors as abstract classes, where arguments to the process constructors are implemented as pure virtual functions and initial values as arguments to the class constructor. For each process, the designer derives from the abstract class implementing the desired process constructor and writes the required virtual functions and then provides the class constructor arguments during instantiation. Then the processes are connected via ForSyDe SystemC channels. Simple SystemC channels implement ForSyDe signals, while more complex channels can implement domain interfaces.

6.4 Foreign Model Integration and Legacy Code

As part of the SYSMODEL project ForSyDe wrappers have been implemented, which allow to integrate foreign models into a formal ForSyDe model as illustrated in Figure 7. These ForSyDe wrappers isolate foreign models from the formal ForSyDe model, but allow the co-simulation of a formal ForSyDe model with a foreign model. The concept of SystemC wrappers does not only allow to include existing models or legacy code into an abstract model, but facilitates also a refinement-by-replacement approach, where processes inside an abstract system model are replaced by low-level implementations, which run on their target platform and are encapsulated in a wrapper. In [2] we have demonstrated how wrappers can be used to integrate C-models running on a target processor, Matlab/Simulink-, and VHDL models and to co-execute these foreign models with the abstract ForSyDe model.

7 Summary

We have presented a system level design methodology based on the ForSyDe formal modeling framework, which allows designers to model and analyze both functional and non-functional properties of their system at the very early stages of the design process. The methodology has been illustrated through the modeling and analysis of a rather simple industry case of a hearing aid calibration device. Early decision support is a very critical factor in handling the design of complex hardware/software systems and to achieve high quality products in short design cycles. We have illustrated how the separation of application and platform in the integrated system model may provide easy explorations of different platforms or mappings. Finally, having a complete and formal system model, may lead to an easier handling of outsourcing sub-parts of the system, as the requirements of the sub-parts may be extracted directly from the system model.

References

[1] ANSI S3.46-1997. Methods of measurement of real-ear performance characteristics of hearing aids, 1997.

[2] S. H. Attarzadeh Niaki and I. Sander. Co-simulation of embedded systems in a heterogeneous MoC-based modeling framework. In *2011 6th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 238–247. IEEE, June 2011.

[3] EN 61669. Electroacoustics - equipment for the measurement of real-ear acoustical characteristics of hearing aids, 2001.

[4] ForSyDe: Formal System Design. `https://forsyde.ict.kth.se/`.

[5] E. Lee and D. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235 – 1245, sept. 1987.

[6] E. A. Lee and A. Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(12):1217–1229, December 1998.

[7] I. Sander and A. Jantsch. System modeling and transformational design refinement in ForSyDe. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(1):17–32, January 2004.

[8] SYSMODEL: System Level Modeling Environment for SMEs. `http://www.sysmodel.eu/`.