A Framework for High-Level Performance Validation of Embedded HW/SW Systems

Dirk Ziegenbein, Rolf Ernst*
Institut fuer Datenverarbeitungsanlagen (Computer Engineering)
Technical University of Braunschweig
Hans-Sommer-Str. 66, D-38106 Braunschweig
{ziegenbein,ernst}@ida.ing.tu-bs.de

Abstract

The specification of an embedded system very often contains several different specification or modeling languages with fundamental differences in their underlying models of computation. This paper focuses on the fundamentals for the joint performance validation of such heterogeneously modeled systems. The main contribution of the presented approach is the coherent internal representation of the different system parts using the SPI model, that is able to capture the important aspects for performance and timing constraint validation for a large number of well-known models.

1 Introduction

With the increasing functionality of embedded systems, the way these systems are designed changes dramatically. Since complex embedded systems include a variety of functions from different domains, they are best modeled with a combination of different modeling languages with fundamental differences in their underlying models of computation. This is not restricted to different description languages for hardware and software parts but e. g. the software part can be described using different modeling languages like data flow graphs or state machines. For a safe integration of the heterogeneously modeled system parts a joint validation is needed. This paper focuses on the joint validation of the performance of such systems and their fulfilment of the imposed timing constraints. Currently, often co-simulation is employed for this purpose but for complex systems the needed complete simulation is not applicable such that successful integration can not be guaranteed.

*This work was supported by the German DFG.

The main problem is a coherent common representation of the differently specified system parts that allows reliable determination of the timing properties of the integrated system. We developed such a common model for the representation of heterogeneously modeled embedded systems [5]. The power of this model lies in the general structure (communicating processes) and the principle of abstraction. The processes are not characterized by their detailed behavior but by parameters that are extracted from the original modeling languages. These parameters include communication behaviors, activation rules and latency times for all possible resources.

Evidently, this common modeling has to include the environment and the imposed constraints. Part of the environment are also possibly used operating systems and their components. The scheduling methods of the operating systems have to be accounted for in performance validation.

2 The Validation Framework

In this section, we will present the main concepts of the proposed framework. First, we will describe the developed high level common design representation the framework is based on. In section 2.2, we will describe the run-time validation tool SYMTA which plays an important role in the parameter extraction. Then the flow of the framework is explained in section 2.3.

2.1 The SPI Model

The core of the proposed validation framework is the common design representation called SPI (**S**ystem **P**roperty **I**ntervals) [5, 6]. In the following, its basic features are reviewed.

The basic model is based on extensions and combinations of classical process models as used in real-time operating systems (RTOS) and synchronous data flow (SDF). It consists of processes communicating via unidirectional channels that are either FIFO-ordered queues (destructive read) or registers (destructive write). The basic model can be represented by a *model graph*.

Definition 1 ***(Model Graph)***
The model graph is a directed bipartite graph $G = (P, C, E)$ *where*

- P *denotes the set of process nodes,*
- C *denotes the set of channel nodes,*
- $E \subseteq (P \times C) \cup (C \times P)$ *denotes the set of edges, and*
- *for each* $c \in C$, $indeg(c) = outdeg(c) = 1$.

The execution of a process instance is based on activation by data availability, i. e. a process is activated if its required input data is present. In section 3 and in [5], it is shown how other activation principles e. g. periodic activation can be modeled by activation by data availability by using virtual model elements, i. e. elements that are introduced for modeling purposes only and do not have to be implemented. The following three points in time give an informal definition of the process execution model:

- *activation time* t_{act}
  required input data is present; process gets activated
- *starting time* t_{start}
  resource is taken; input data is read; process starts execution
- *completion time* t_{comp}
  input data is consumed (i. e. destroyed); output data is written; resource is released; process execution is completed.

Note that the process execution's effect becomes visible on the channels as one atomic action at the end of execution. Since communication may also consume time, the time a token is put on a channel (t_{comp} of writing process) need not equal the time this token is available for being read and, thus, for activating the succeeding process. Therefore, this time is defined as the *output availability time* t_{av}.

The properties of the processes and channels are defined by parameters annotated to the corresponding graph elements. This way, the model can be easily adapted to include all required information for a certain task in the design flow. The parameters need not be fixed but can be specified using uncertainty intervals, i. e. they are constrained by an upper and lower bound. The sources for this non-determinism can be data dependent functionality of a process or an incomplete specification. This also enables estimation of system parts in early design stages and the integration of legacy code in the limits of its formal analyzability.

For high-level performance analysis of a system of communicating processes, the detailed functionality of the processes is not needed. It is sufficient to know for each process the resource requirements and the timed interaction with its environment. A key point in this context is the amount of data to be communicated. It is needed to determine the process activations and to analyze the sufficiency of the available bandwidth of communication resources. Therefore, a *data rate* denoting the number of data tokens communicated at a process execution is specified. Together with the data size of a token, the absolute amount of communicated data can be easily calculated.

Definition 2 ***(Data Rates)***
Let $Inputs(p) = \{c \in C \mid e = (c,p) \in E\}$ *denote the set of input channels of process* $p \in P$ *and* $Outputs(p) = \{c \in C \mid e = (p,c) \in E\}$ *denote the set of output channels of* p. *Associated with each process node* $p \in P$ *and each input channel* $c \in Inputs(p)$, *there is an input data rate* $r_c(k)$ *that denotes the number of data tokens the process* p *consumes from the channel* c *at its* k*th execution. This rate* $r_c(k)$ *is constrained by an interval* $R_c = [r_{c,min}, r_{c,max}]$, *such that* $\forall k : r_c(k) \in R_c$. *Analogously for each output channel* $c \in Outputs(p)$, *there is an output data rate* $s_c(k)$ *and a constraining interval* $S_c = [s_{c,min}, s_{c,max}]$.

Note that the interval representation allows processes to have non-constant data rates. Contrary to existing data flow models (e.g., synchronous data flow [1]) where data rates are fixed and time-invariant, our representation allows for the modeling of non-determinism to abstractly capture conditional behavior.

Furthermore, we need to define some constructs to keep track of the tokens and to model the availability of data.

Definition 3 ***(Data on Channels)***
Associated with each channel $c \in C$, *there are the numbers* $\hat{d}_c$, d_c, *and* $d_{c,av}$ *where*

- $\hat{d}_c$ *denotes the initial number of data tokens,*
- d_c *denotes the total number of data tokens at a given point of time,*
- $d_{c,av}$ *denotes the number of data tokens available for activation of the succeeding process at*

a given point of time [1]

on channel c. Note that all three numbers may be uncertain.

The central point for performance analysis is the timing of the system components. Therefore, latency times are defined for all processes and channels.

Definition 4 ***(Latency Times)***
Associated with each process $p \in P$, there is a latency time $lat_p(k) \in Lat_p = [lat_{p,min}, lat_{p,max}]$ where $lat_{p,min}$ [$lat_{p,max}$] denotes the lower [upper] bound on the execution time ($t_{comp,p}(k) - t_{start,p}(k)$) of instance k of process p.

Analogously, associated with each channel $c \in C$, there is a latency time $lat_c(k) \in Lat_c = [lat_{c,min}, lat_{c,max}]$ that limits the communication time ($t_{av,p}(k) - t_{comp,p}(k)$ where p writes on channel c) for a token on channel c.

Note that the interval representation of the latency times matches the output of our timing analysis tool: best and worst case execution times, both of which are needed for safe performance validation. Of course, latency times are resource dependent but for the task of performance validation we assume a static mapping of processes and channels to resources.

With the above presented annotations, a process in the SPI model has a single behavior that is obtained by abstraction of possibly several different execution paths in the original process and represented using non-determinism bounded by uncertainty intervals. In some cases, abstraction may lead to too imprecise modelings with not enough detail for efficient scheduling or successful validation. Therefore, the concept of process modes was added to the SPI model. A process mode describes a subset of the possible process behaviors. To correlate the different process modes, mode tags are introduced that model information contained in the communicated data but not yet exposed with the previous set of annotations. When a process receives a mode tag it adapts its mode according to the value of the mode tag and possibly transmits the mode tag or generates a new mode tag on its outputs. For a more formal description of modes and some examples see [6].

For the validation of embedded systems, the environment and the constraints that are imposed on the system have to be modeled as well. Using virtual model elements, this can be done in a way that is coherent with the modeling of the system. Virtual model elements that have the same properties as the ones that are used to model the embedded system but do not have to be implemented.

Timing constraints can be specified for paths of channels and restrict the time tokens may spend on these paths.

Definition 5 ***(Latency Path Constraints)***
A path constraint is a labeled path in the model graph. A path is of the form $(P_1 \longrightarrow C_1 \longrightarrow P_2 ... \longrightarrow C_n \longrightarrow P_{n+1})$ while involving $n+1$ processes, n channel nodes and $2n$ edges. As each channel uniquely identifies its sender and its receiver process, we describe a path more shortly by an expression of the form

$$path = (c_1, \cdots, c_n)$$

where $c_1, \cdots, c_n \in C$. A latency path constraint

$$LC_{path} = [t_{lat,min}, t_{lat,max}]$$

denotes the time interval which tokens are allowed to travel along the specified path from C_1 to C_n. The path constraint must be satisfied for any token during any possible execution of the system.

For a more detailed definition of latency path constraints see [6]. Using virtual model elements, other types of timing constraints like periodicity constraints can be modeled using latency constraints (e. g. process P_{time} in Figure 3 is constrained to be executed at an exact rate of t_1).

It has been shown that the SPI model is capable to capture the important aspects for performance validation from different data flow models (static data flow, boolean data flow, ...), hardware description languages (VHDL), underlying process models of real time operating systems (periodic processes) and state-based models (StateCharts). Due to its dynamic process instantiations, SDL, an important model from the telecommunications domain can be supported only partially. The user has to add a modeling of the environment including timing constraints and processes modeling signal sources and sinks and operating system components.

2.2 SYMTA

The parameter extraction for the mapping to the SPI model is based on SYMTA (**SYM**bolic hybrid **T**iming **A**nalysis) [4], a timing analysis tool that combines formal analysis and simulation. The input representation is a control and data flow graph (CDFG) that is currently derived from the C language but can be easily adapted to all imperative programming languages. Based on the CDFG and possible user annotations, the program is segmented and

[1] Note that $d_{c,av} \leq d_c$ since $d_{c,av}$ is the already communicated part of the total amount of data d_c. If no latency is associated with the channel then $d_{c,av} = d_c$.

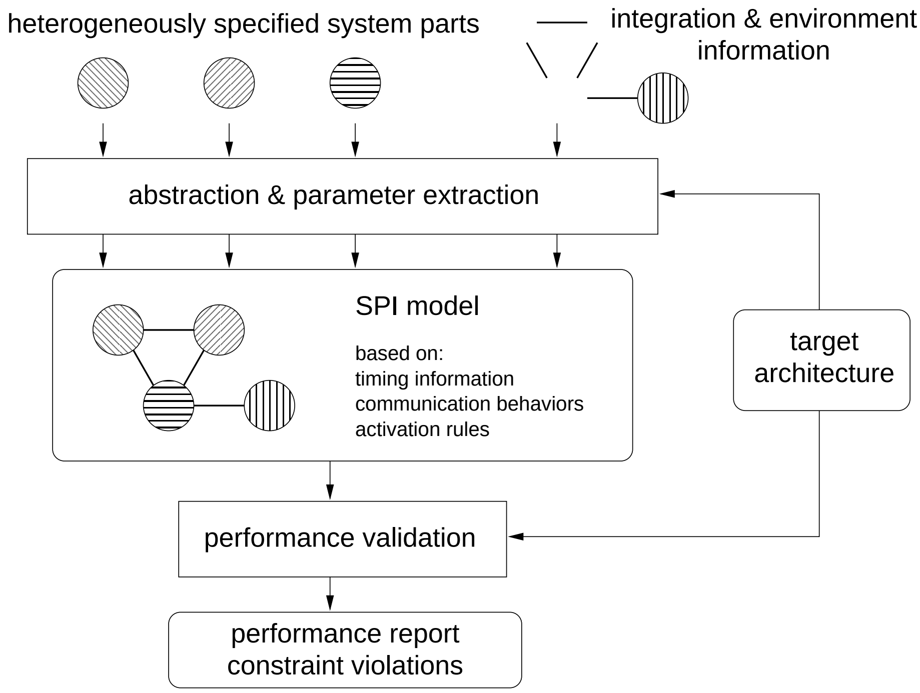

Figure 1: The Validation Framework

classified into single feasible path (SFP) and multiple feasible path (MFP) segments. These properties depend on the existence of a single or multiple control paths through the program segment. Similarly, the target processor architecture is classified using characteristics like pipelining, caches, parallel execution and data-dependent instruction execution times. Based on these classifications, the analysis technique with the tightest bounds on the latency time of each program segment are safely selected.

The data flow analysis and the symbolic execution performed by SYMTA also yields execution number intervals for each operation in the source code. From this, the amount of communicated data and thus also the data rates and the activation rule can easily be derived.

2.3 Performance Validation

The flow of our proposed approach can be seen in Figure 1. The first step is parameter extraction. Here, the parameters needed for the SPI model are obtained. Since the latency time is one of these parameters, the intra-process timing analysis performed by SYMTA is part of this step. Due to the resource dependency of latency times, the resources have to be known at this step. If the validation framework is part of an iterative scheduling and partitioning loop, the parameter extraction is also possible for a set of possible resources.

Using the concept of process modes, the degree of abstraction used in parameter extraction can be controlled. The extreme cases are specifying a mode for each possible execution path of a process (high accuracy of modeling but exponential growth of paths with number of branches) or specifying a single behavior using uncertainty intervals (low accuracy but problem size reduction). Thus, during parameter extraction, a trade off between problem size and accuracy of modeling which is directly related to the accuracy of the results is possible. It has to be stressed that what is called lower accuracy of modeling does not lead to a false acceptance of a system since all cases (execution paths) are included in the uncertainty intervals. On the other hand, false rejections are possible since the uncertainty intervals may also include cases that are not part of the specification.

Another advantage of our approach is that due to the abstraction only the structural implementation details have to be disclosed but all functional implementation details can be hidden. This ”grey box” view is particularly useful when validating performance for systems that are designed in a multi-company effort. An example for this is the design of automotive electronics where the car manufacturer and its subcontractors have to commonly validate the system without giving away too many implementation details which may hurt their competing business interests.

The next step is the integration of the different system parts using the SPI model. Additional model elements representing the environment and operating system components are added at this stage. Again, it has to be stressed that SPI is not used as a new modeling language but as a common internal representation for the needed information obtained from the different original modeling or specification languages. Starting from the system description in the SPI model, now a coherent validation of the system's performance including the verification of the timing constraints can be performed.

We are currently planning to develop a publicly available SPI workbench which will include an open extensible data structure representing the SPI model, extractors for several modeling or specification languages, methods for scheduling, allocation and verification and a user interface that will allow the definition of subclasses of the SPI model and easy integration of model extensions or new methods. The user will be able to perform experiments by embedding his own methods into the infrastructure provided by the workbench.

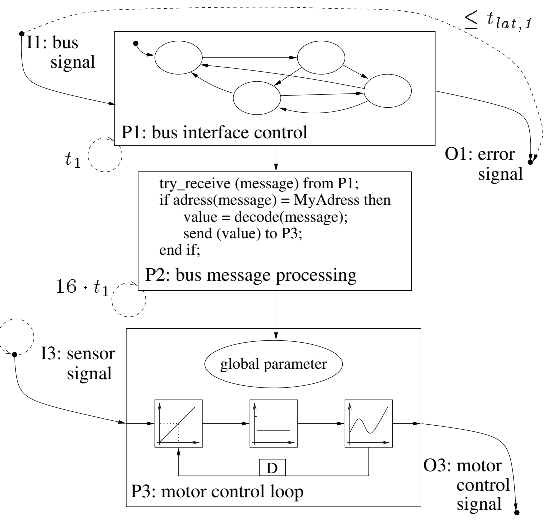

Figure 2: Remote Motor Controller

3 Example

In this section, we will demonstrate our approach using an example of a remote motor controller. The system collects message parts from a bus and tests them for an error (P_1), decodes the collected message (P_2) and sends a control word to the motor control loop (P_3). A description of the system, in which process P_1 is specified in a state-based language, process P_2 in a C derivative and process P_3 in a static data flow language, is depicted in Fig. 2. The interaction of the processes and the environment is loosely defined. P_1 and P_2 are both periodic processes while P_3 is driven by a periodic input. There is a timing constraint that limits the maximum response time to an erroneously received message part.

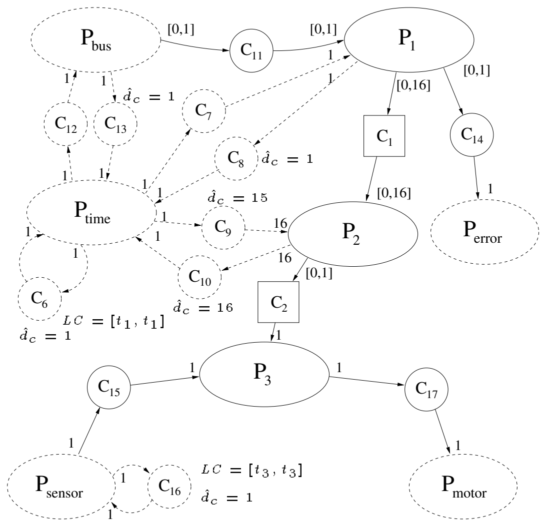

Figure 3: Remote Motor Controller (our representation)

A mapping of the system to the SPI model is depicted in 3. The communication between the processes is mapped to the registers C_1 and C_2 (denoted by rectangles). The response time constraint is mapped to the latency path constraint $LC_{(C_{11}, C_{14})} = [0, 0.8\, t_1]$. Note that this is just one possible mapping from the original description to the SPI model as e. g. one could have also mapped the SDF to three different processes.

Figure 3 also serves as a good example for the modeling of different activation principles by activation on data availability. An example for the activation by relative execution rates (e. g. RMS semantics: no exact periodicity but constrained mobility intervals [2]) is process P_2 which has to be executed once during every period $16 \cdot t_1$ and, thus, once during 16 executions of P_{time}. This is modeled by two virtual channels (C_9 and C_{10}) with preassigned tokens between both processes. Thus, with its first execution P_{time} enables one activation of P_2 whose execution enables another 16 activations of P_{time} that lead to another activation of P_2 etc.

Assume that the processes in Figure 3 are all mapped to the same processor. For this case, the latency times of the processes can be seen in the first column of Table 1. The virtual processes, represented by dotted lines, are assumed to have a zero latency.

Process	Latency	C_i	T_i	D_i
P_1	$[0.2, 0.3]\,t_1$	$0.3\,t_1$	t_1	$0.8\,t_1$
P_2	$[0.6, 1]\,t_1$	t_1	$16\,t_1$	$16\,t_1$
P_3	$[1.1, 1.2]\,t_1$	$1.2\,t_1$	$8\,t_1$	$8\,t_1$

Table 1: Latency Time Intervals and GRMS Parameters

To give a simple example how the performance of a system modeled in the SPI model can be validated, we apply a generalized rate monotonic scheduling analysis method [3] to verify if the system is schedulable under the given timing constraints. This method can be applied and the processes can be seen as independent since their communication is non-blocking. The latency path constraint over process P_1 can be accounted for as deadline D_1 while for the other processes the deadlines equal their periods ($D_i = T_i$). The GRMS method only uses the worst case execution times C_i which can be found with the other parameters ($t_3 = 8\,t_1$ assumed) in Table 1. With

$$\forall i, 1 \leq i \leq n, \frac{C_1}{T_1} + \cdots + \frac{(C_i + E_i)}{T_i} \leq i(2^{1/i} - 1)$$

$$\textit{where} \quad E_i = T_i - D_i \quad \textit{and} \quad D_i \leq D_{i+1}$$

and

$$\frac{(C_1 + T_1 - D_1)}{T_1} = 0.3 + 1 - 0.8 = 0.5 \leq 1$$

$$\frac{C_1}{T_1} + \frac{C_3}{T_3} = 0.3 + 1.2/8 = 0.45 \leq 0.83$$

$$\frac{C_1}{T_1} + \frac{C_3}{T_3} + \frac{C_2}{T_2} = 0.3 + 1.2/8 + 1/16 = 0.51 \leq 0.78$$

it is shown that the timing constraints can be fulfilled. For reasons of simplicity, possible operating system components are not taken into account.

4 Conclusion

We proposed a framework for a coherent performance validation of heterogeneously specified embedded systems. The core of our approach is the SPI model, a common representation of the different system parts in a single model to create a coherent starting point for validation algorithms.

References

[1] E. A. Lee and D.G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on Computers*, 36(1), January 1987.

[2] C. Liu and J. Layland. Scheduling algorithm for multiprogramming in a hard-real-time environment. *Journal of the ACM*, pages 46–61, 1973.

[3] L. Sha, R. Rajkumar, and S. S. Sathaye. Generalized rate monotonic scheduling theory: A framework for developing real-time systems. *Proceedings of the IEEE*, 82(1):68–82, January 1994.

[4] W. Ye and R. Ernst. Embedded program timing analysis based on path clustering and architecture classification. In *Proceedings International Conference on Computer-Aided Design (ICCAD '97)*, San Jose, USA, 1997.

[5] D. Ziegenbein, R. Ernst, K. Richter, J. Teich, and L. Thiele. Combining multiple models of computation for scheduling and allocation. In *Proceedings Sixth International Workshop on Hardware/Software Co-Design (Codes/CASHE '98)*, pages 9–13, Seattle, USA, 1998.

[6] D. Ziegenbein, K. Richter, R. Ernst, J. Teich, and L. Thiele. Representation of process mode correlation for scheduling. In *Proceedings International Conference on Computer-Aided Design (ICCAD '98)*, San Jose, USA, November 1998.