ELEKTROTEHNIŠKI VESTNIK 78(4): 165–170, 2011
ENGLISH EDITION

A Design Analysis Tool for Embedded Systems Based on Abstract System Modelling

Klemen Perko[1], **Tomaž Nahtigal**[2], **Andrej Trost**[2]

[1]*Sipronika d.o.o., Tržaška cesta 2, 1000 Ljubljana, Slovenija*
[2]*University of Ljubljana, Faculty of electricale Engineering, Tržaška 25, SI-1000 Ljubljana, Slovenia*

E-mail: andrej.trost@fe.uni-lj.si

Abstract. Design analysis and evaluation are an important step in the embedded system design process. The accuracy of the evaluation results greatly affects the decisions in the design process. The paper presents ASyMod, a tool to be used in evaluation of embedded systems in the early steps of the design process. The tool enables graphical composition of the system models. The embedded system models are composed of hardware and software components described on an abstract system level. The abstract models can be quickly assembled and evaluated according to the hardware usage and task execution timing. We will present an example of embedded system modelling and evaluation in the ASyMod tool. We will show that the abstract-model analysis results match with measurements on a real embedded system.

Keywords: embedded systems, hardware and software modelling, abstraction

1 Introduction

Embedded systems are electronic systems that perform a specific function like control, regulation or data processing. Contemporary embedded systems are heterogeneous systems, consisting of multiple processing units connected with communication busses. A typical embedded system performs a number of dedicated functions that can be divided into tasks. Some tasks are handled by dedicated hardware components, an example is a communication interface circuit, but usually most of the tasks are implemented as a program running on a microprocessor [1]. An operating system is used for scheduling the tasks on the processor, seemingly making them run in parallel from the systems point of view.

Designing contemporary embedded systems poses a challenge due to their increasing complexity. New design methodologies are required in order to meet the demands for high quality, low price and quick time-to-market. The designers need modelling tools that enable an early evaluation of the embedded system which will effectively guide the decisions in the design process. They need data about task execution times and hardware resource usages in the early stages of the design process [2]. Many embedded systems are reactive systems with strict timing requirements that have to be taken into account. For example, during design of an embedded control system, data sampling and actuation timing periods are set. Deviations from these periods lead to worsening of regulation performances.

System Level Design (SLD) methodologies rely on modelling as one of the basic concepts [3], [4]. The design process starts with development of the high-level system model [5], which has no details about implementation. The high level or abstract model is evaluated according to requirements stated in the specification. If the specification requirements are not fulfilled, the model has to be changed in the design space exploration phase [6]. When the requirements are met, the model is refined with further details and re-evaluated. The model refinement and evaluation cycle is repeated until it contains all the implementation information.

The paper presents a tool for modelling and evaluation of embedded systems on a very high abstraction level. First, we define the basic components of the system model, then the design composition and evaluation is explained. The design process is further revealed in a case study. In the results section, we show a good match between the model evaluation and measurements on a real embedded system.

2 Abstract System Model

Our modelling methodology allows for hardware and software co-design [7]. It is based on the most commonly used set of abstractions that defines the minimal amount of data needed for modelling and evaluation of embedded reactive systems with distributed data processing. Reactive systems rely on a deterministic task execution times, which depend on the speed and usage

Received October 19, 2011
Accepted November 7, 2011

of hardware execution units, for example processors, and communication delays.

The embedded system design process starts with the specification and set of known algorithms that will be implemented either in hardware, software or a mixture of both. The specification influences the design decisions. Using algorithms already implemented in a software is only one of the possible implementations and not necessarily optimal for the system being designed.

The system model can be divided according to the Y graph concept [8] into *functionality* or algorithm description, *architecture* or hardware description and *mapping*.

In the functionality model, we define the system behaviour as a network of abstract tasks connected by their data dependencies. The abstract task is described as a sequence of high-level data processing operations and data transfer requests. The abstract architecture model contains a set of execution units and communication units. Execution units provide resources for data processing. In the abstract model, only an estimation of the execution delay for a specific operations is provided. Communication units are used for modelling access to the system buses. Several execution units can share one communication unit, but only one execution unit at a time is able to use the communication unit while the others are in a waiting state.

The tasks are assigned to the execution units in the mapping model. The purpose of mapping is to specify which execution unit will serve the requests of a specific task. The task model can be seen as a sequence of data processing and transfer requests. The requests are served according to the availability of a particular execution unit and corresponding communication unit during the model simulation phase.

The execution units are implemented as dedicated hardware components or general-purpose processing components, for example microprocessors. Dedicated components usually process only a small number of requests, typically only one operation. In such a case the assigned task has an exclusive access to the component. On the other hand, general processing components serve several tasks handled by the operating system scheduler. The abstract model of the general processing component includes task scheduling. We define several states for the task:

- Wait - the task is waiting for a trigger event or communication unit,
- Ready - the task is ready for execution, and
- Run - the task is currenty being executed.

The scheduler periodically checks a list of tasks and selects the one ready for execution. A selected task changes its state from Ready to Run and starts execution of operations. Selection depends on the task attributes and the scheduling algorithm. When the task is finished, the scheduler removes it from the list. Our scheduler model can implement several typical scheduling algorithms used in embedded systems [9]: First-Come First-Served (FCFS), Rate Monotonic (RM), Deadline Monotonic (DM) and Earliest Deadline First (EDF). The designer can also choose between non-preemptive and preemptive scheduling, where the high priority tasks interrupt execution of the lower priority tasks.

3 Abstract-System Modeling Tool - ASyMod

We designed a new tool for embedded system modelling and evaluation on a high abstraction level. The tool called ASyMod (Abstract System Modelling) is composed of a graphical modeling environment with support libraries, an interpreter and model evaluation scripts.

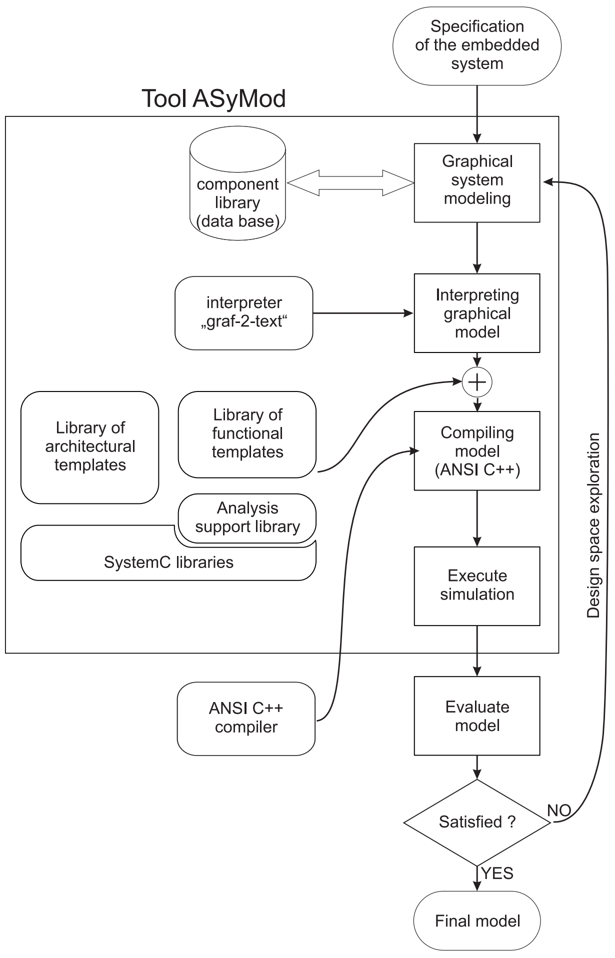
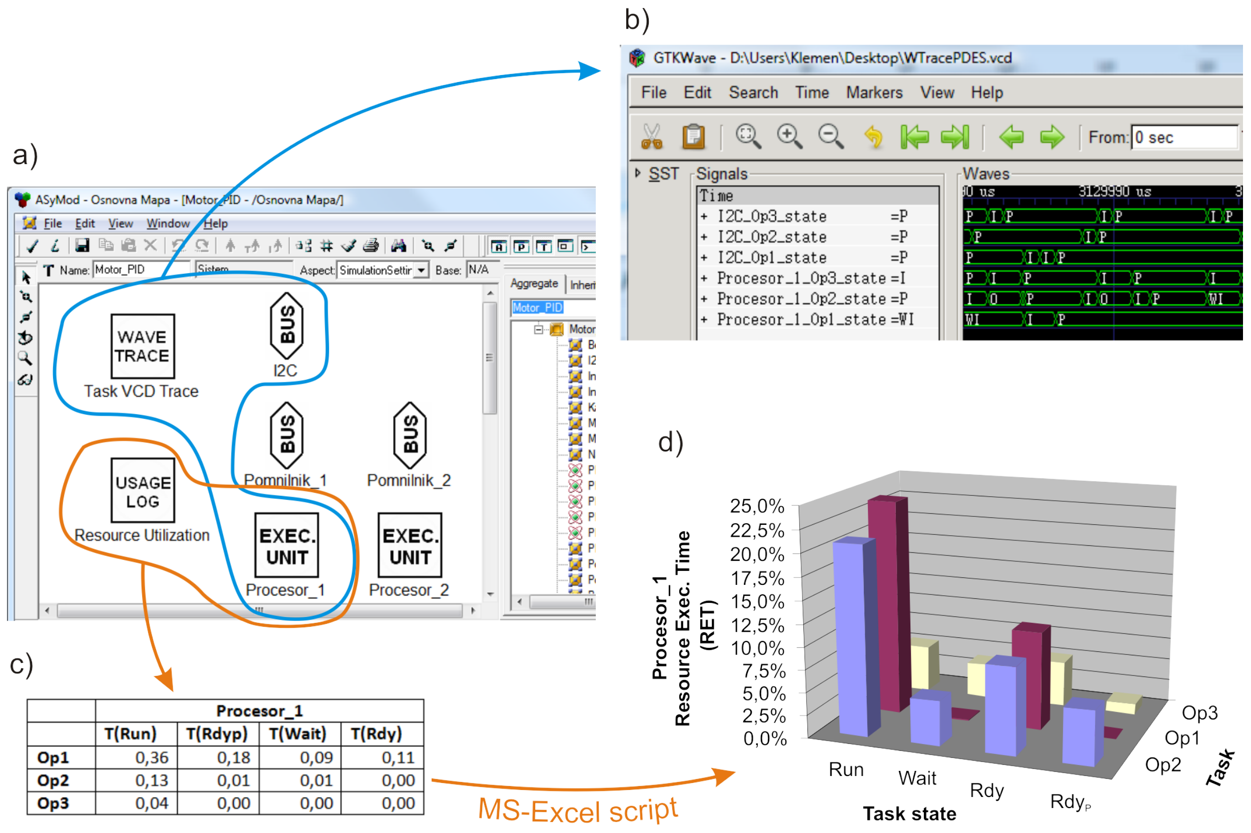

Specification of the embedded system
Tool ASyMod
component library (data base)
Graphical system modeling
interpreter „graf-2-text“
Interpreting graphical model
Library of architectural templates
Library of functional templates
Compiling model (ANSI C++)
Analysis support library
SystemC libraries
Execute simulation
Design space exploration
ANSI C++ compiler
Evaluate model
Satisfied ?
NO
YES
Final model

Figure 1. Embedded system modelling in the developed ASyMod tool

Fig. 1 presents the ASyMod components and embedded system modelling flow. The model is built using the existing components in the graphical environment. The graphical modelling environment is based on a



```
Time
+ I2C_Op3_state        =P
+ I2C_Op2_state        =P
+ I2C_Op1_state        =P
+ Procesor_1_Op3_state =I
+ Procesor_1_Op2_state =P
+ Procesor_1_Op1_state =WI
```

	Procesor_1			
	T(Run)	T(Rdyp)	T(Wait)	T(Rdy)
Op1	0,36	0,18	0,09	0,11
Op2	0,13	0,01	0,01	0,00
Op3	0,04	0,00	0,00	0,00

Figure 2. Model evaluation in the ASyMod flow: a) graphical modelling, b) simulation traces, c) resouce-usage statistics and d) task-state graphs

configurable toolkit GME (Generic Modeling Environment) [10]. We provide a metamodel with all syntactical, semantic and presentation information for the abstract system models. This metamodel is used to configure GME for a specific embedded systems domain.

Graphical modelling takes place in three aspects: *architectue*, *functionality* and *mapping* following the Y graph methodology. In the first two aspects, the system model components are defined. The designer can use one of the existing library components or design a new component and save it in the library for future use. The components must conform to the modelling semantics. The graphical environment guides the designer during component and model creation and instantly checks the design rules. In the mapping aspect, the designer connects functional components with architectural execution units.

After the graphical modelling phase, the system model is converted using a system modelling language. An interpreter takes the graphical model and component properties and produces code in the system modelling language SystemC. The code consists of the abstract system model, the test bench and the simulation setup. The setup defines the observed components, required outputs, simulation length and time steps.

The next step is compilation of the code with a standard ANSI C++ compiler producing an executable file. By running the file, the simulation is performed, and results are collected, as shown in Fig. 2. Simulation results are presented as timing traces. Besides that, a statistical report containing the usage of resources is prepared. A script in MS-Excell can be used for a graphical view of the observed tasks states.

The simulation traces are difficult to evaluate and better system performance metrics can be obtained by statistical interpretation of the traces. The ASyMod tool generates a table report for each scheduled task. The report provides relative portions of the total waiting, ready, preempted and running time for the task. The task running periods and their variations (jitter) are important for performance evaluation of reactive systems and are collected, interpreted and presented in graphs. Our simulation results will be shown in the case study section below.

The embedded system design is an iterative process. If the initial model is not satisfactory, the designer should change and re-evaluate the model. This process of the design space exploration is repeated, until the evaluation results satisfy the specification requirements.

4 Embedded System Modelling Case Study

System level modelling in the ASyMod tool will be presented on an example of a motor speed-control system. Functionality of the system is based on the discrete PI control algorithm. The algorithm periodically executes three data-dependent tasks: reading speed sensor (`Get`), computing response (`PI`) and setting the actuator (`Set`), as presented in Fig. 3. The embedded system is often responsible for other tasks which are modelled by two periodic tasks: `T2` and `T3`. These tasks are not directly data-dependent on the control task, but they share the same execution unit (processor) and communication bus. Due to resource sharing, we expect some additional delay which disturbs the control operation and the purpose of modelling is to evaluate the delay before the actual implementation.

Resource sharing is handled by the operating system with a task scheduler. The scheduler periodically runs three scheduled tasks: `Op1`, `Op2` and `Op3`, with trigger periods defined in components `PEG1`, `PEG2` and `PEG3`, as shown in Fig. 3. Among different possible scheduling algorithms, we select the rate monotonic preemptive scheduler.

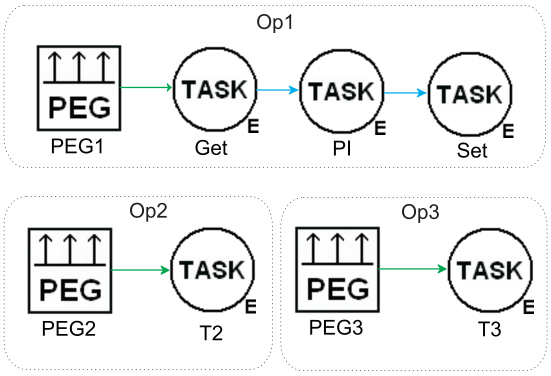


Figure 3. Functional aspect of the developed embedded control system in ASyMod

The architecture model consists of one execution unit `Exec_ARM7` and two communication units: `Mem` and `Comm`. The first two units present the processor with a local memory. The unit `Comm` is a model of a serial communication bus between the processing unit and motor speed sensor. Fig. 4 presents the mapping aspect of the ASyMod embedded system model.

A snippet of the SystemC code is given in Fig. 5. The code describes tasks `Get` and `PI` and a corresponding part of the execution unit ARM7. Description of the task `Get` consists of a request for a two-byte data transfer from the communication unit `Comm`.

The model requires estimation of the data transfer delay on the bus and the task execution delay on the processor. Estimations can be obtained by a low-level

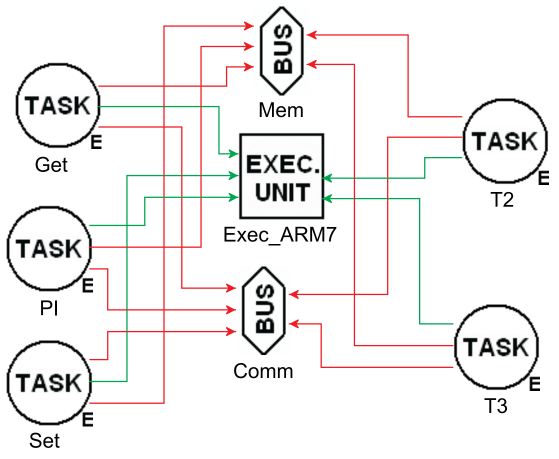


Figure 4. Mapping aspect of the model in ASyMod

```
// Abstract task description in ASyMod
void Get::MainThread()
{
 m_pExecUnit->GetData(this, 2, &Comm, -1);
}

void PI::MainThread()
{
 m_pExecUnit->Calc(this, 563);
}

// ARM7 library services
void Exec_ARM7::Calc(int n)
{
  for (int i=0; i<563; i++)
  {
    Wait(this, sc_time(64, SC_NS));
  }
}
```

Figure 5. High-level model in SystemC

simulator or measured on the implemented system. Once the estimations for a certain technology are collected, they can be saved in the library for future use. The discrete control algorithm delay estimate is 36 µs or 563 machine instructions. Since the model assumes preemptive execution, the task delay is modelled with a loop that can be interrupted at any time by the scheduler. The unit delay in a loop of 64 ns is an average delay of the machine instructions.

5 Results

In Fig. 6 we define some characteristic times of the periodic task execution:

1) T_k: time between the k-th run and $(k+1)$-th run of the task
2) $trig2end_k$: time between trigger and end of task execution,
3) $trig2start_k$: delay from the request for execution (trigger) to the actual run state,

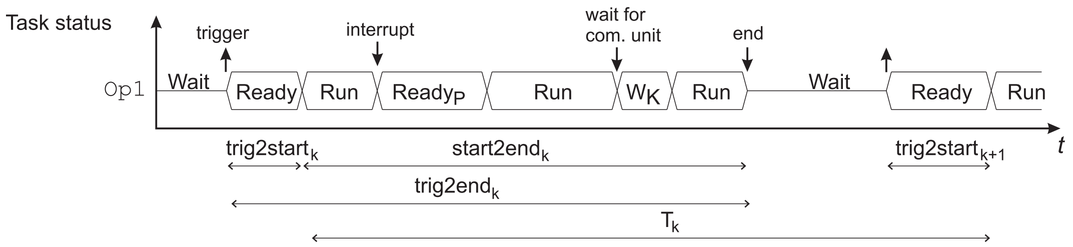


Figure 6. Characteristic times for periodic task execution.

4) $start2end_k$: cumulative time of task execution including all interruption.

Characteristic times are sampled during model simulation, statistically processed and presented in a histogram.

Fig. 7 shows normalized histograms of two characteristic times for the task `Op1`. The sample population contains 313 measurements divided into 22 columns. The scheduled task `Op1` has the lowest priority. The delay from the trigger to the actual running state $trig2start$ has the most spread distribution. The probability of an immediate execution of this task after the trigger is slightly over $p_1 = 0.2$, so it is very likely that it will need to wait for the higher-priority scheduled tasks to finish.

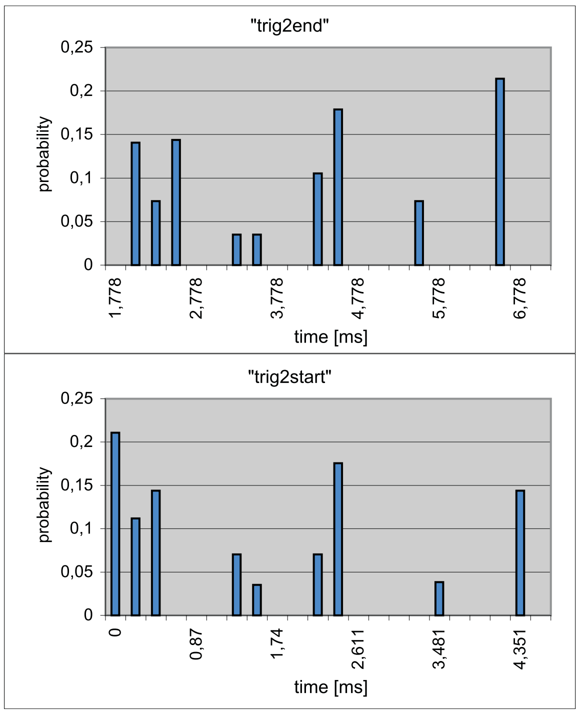


Figure 7. Histograms of characteristic times for the task `Op1`

To allow a comparison, we measure the characteristic times on the implemented system. The code is modified with test instructions for signalling the task state. When a task starts running, a logic 1 is outputed on a microcontroller port. The port changes to 0 when the task is finished. The measurements are sampled with a digital storage oscilloscope Lecroy WaveRunner and a reference signal generator. Functions of the oscilloscope provide continuous data sampling, processing and displaying the results in histograms presented in Fig. 8.

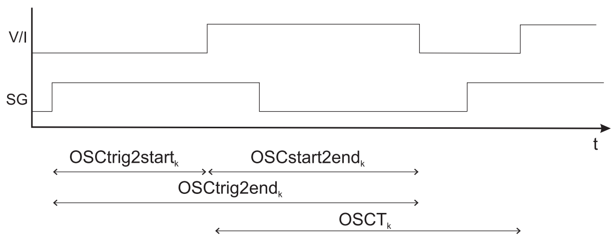


Figure 8. Oscilloscope measurements of the characteristic timings.

The measured pulse widths $OSCstart2end_k$ are equivalent to the characteristic time $start2end$. $OSCT_k$ is the measured time between the rising edges on the port and is equivalent to the period T. The reference signal generator is used for measurement of characteristic times $trig2start$ and $trig2end$. The generator is set to the frequency corresponding to the particular task period. The rising edge is used for triggering the oscilloscope that samples the logic value on the microcontroller port. The measured delay $OSCtrig2start_k$ between the trigger and the rising edge of the port signal is correlated to the time $trig2start$. Similarly $OSCtrig2end_k$ is the delay between the trigger and the falling edge of the port signal correlated to the time $trig2end$.

The measurement of $OSCtrig2start$ and $OSCtrig2end$ has an unknown phase delay between the actual task trigger condition and the generated trigger signal. This provides an additional time shift according to the actual timings $trig2start$ and $trig2end$ that cannot be systematically corrected with this measurement method. But the time shift does not change the shape of the measured histogram (column heights and relative column distances), the impact is only in the horizontal shift. Each column is equally shifted for the phase delay value. Fig. 9 presents the measured histogram of timing $OSCtrig2end$ and $OSCtrig2start$. A quick comparison shows similarity with the simulated results from Fig. 7.

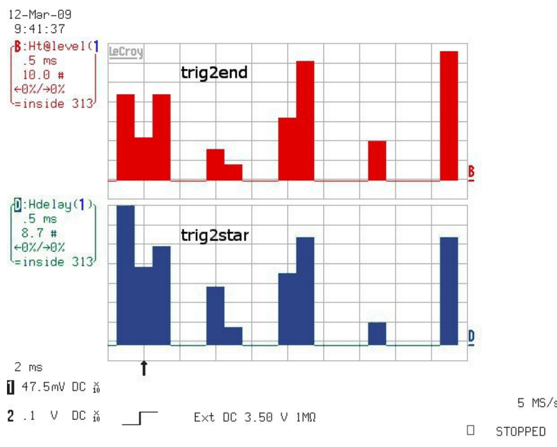


Figure 9. Measured histograms characteristic times $OSCtrig2start$ and $OSCtrig2end$ for the task `OP1`.

The histograms can be quantitatively compared by the method of normalized histogram interception. The similarity between the histograms is defined as the intersect area of two normalized histograms. In the histograms with B bins, minimum probability for each pair of bins is summed up:

$$SIM = \sum_{i=1}^{B} min(p_i, p_{Ri}) \quad . \tag{1}$$

The result SIM lies between 0 and 1. If the value is closer to 1, the histograms are more similar. Comparison of simulated $trig2end$ and measured $OSCtrig2end$ gives $SIM = 0.97$, and comparison of $trig2start$ and $OSCtrig2start$ gives $SIM = 0.99$.

6 Conclusion

The ASyMod tool offers a good base for further research in the area of the system design. The results of our abstract-model simulation will be useful in our future exploring of the design space and of the options for enabling system model refinement.

Currently, the developed tool supports modeling of the delay and resource usage, irrespective of the actual data dependency on the delay. In our next refinement stage, the data dependency can also be included. The operating system model can be improved by additional information about the delay for the context switch as well.

The communication units currently use a mutex concept, where the unit is either available or busy and the access delay is modeled as part of the functionality. The communication refinement could use a model describing the communication channel with a particular defined bandwidth. The functional unit would request a service from the channel and computation of the delay would be performed in the channel model. By doing so, separation between embedded system functionality and resources provided by the architectural components would be improved.

References

[1] Frank Vahid and Tony Givargis, "Embedded System Design: A Unified Hardware/Software Introduction", John Wiley and Sons, 2002.

[2] J. Kreku, M. Hoppari, T. Kestilä, Y. Qu, J.-P. Soininen, P. Andersson and K. Tiensyrjä, "Combining UML2 application and SystemC platform modelling for performance evaluation of real-time embedded systems", EURASIP Journal on Embedded Systems, vol. 2008, no. 6, pp. 1-–18, 2008.

[3] V. Zivkovic and P. Lieverse, "An Overview of Methodologies and Tools in the Field of System-Level Design", Embedded Processor Design Challenges, Springer-Verlag New York, Inc., pp. 74—88, 2002.

[4] Alberto Sangiovanni-Vincentelli, "Quo Vadis SLD: Reasoning about Trends and Challenges of System-Level Design", *Proceedings of the IEEE*, vol. 95, no. 3, pp. 467—506, March 2007.

[5] D.C. Schmidt, "Guest editor's introduction: model-driven engineering", IEEE Computer, vol. 39, no. 2, pp. 25-–31, 2006.

[6] A. Pimentel, "The Artemis Workbench for System-level Performance Evaluation of Embedded Systems", Journal of Embedded Systems, vol. 3, no. 3, pp. 181-–196, 2008.

[7] J. Dedič, M. Finc, A. Trost, "A methodology for supporting system-level design space exploration at higher levels of abstraction", J. Circ. Syst. Comput., vol. 17, no. 4, pp. 703—727, 2008.

[8] Daniel D. Gajski and Robert H. Kuhn, "New VLSI tools" *Computer*, vol. 16, no. 12, pp. 11—14, 1983.

[9] G. Buttazzo, "Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications", Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[10] A. Ledeczi et all, "The Generic Modeling Environment", Proceedings of IEEE Int. Workshop on Intelligent Signal Processing (WISP '01), IEEE, Budapest, Hungary, maj 2001.

Klemen Perko received his B.Sc. and Ph.D. degrees in electrical engineering from the Faculty of Electrical Engineering, University of Ljubljana, Slovenia in 2004 and 2011, perspectively. Since 2005 he has been engaged with the company Sipronika and has been working on several applicative projects. His research interests include high-level design and modeling of embedded systems.

Tomaž Nahtigal graduated from the Faculty of Electrical Engineering, University of Ljubljana in 2007. He is now studying towards his Ph.D. degree. His research interests include design and verification of digital systems, applications and hardware software co-design.

Andrej Trost received his Ph.D. degree in 2000 from the Faculty of Electrical Engineering, University of Ljubljana. Currently he works at the same faculty as an assistant professor teaching high-level design techniques on several graduate and post-graduate study levels. His research interests include the FPGA technology and digital systems design for academic and industrial applications.