C-based System Development of Asynchronous Distributed Systems*

Mario Korte
Department of Computer Science
University of Oldenburg
Ammerländer Heerstr. 114, 26111 Oldenburg
Mario.Korte@informatik.uni-oldenburg.de

Frank Slomka
Department of Embedded / Real-Time Systems
University of Ulm
Albert-Einstein-Allee 11, 89069 Ulm
Frank.Slomka@uni-ulm.de

Abstract

Embedded control systems have to be functionally correct, stable, and have to fulfil real-time constraints. The presented integrated approach of embedded software and hardware development supports the developer to meet design decisions in a system context using simulations at the system level and the task level of modelling. At the system level, first the behaviour of the digital controller model is specified. Secondly, the developer explores at the task level various alternatives of task scheduling policies and hardware/software architectures using simulations based on task models and virtual prototype models of the hardware. Estimated delays and scheduling times are back-annotated to the system level model. There the annotations are used to correct the digital control algorithm parameters that the embedded system remains functionally correct and stable.

1. Introduction

Embedded systems are designed to do some specific task such as engine control in cars, communication devices, and household appliances. Especially systems in automotive and industrial control applications are constrained by real-time conditions. Their violation may lead to a system failure. New technologies allow creating complex embedded systems that consist of heterogeneous subcomponents such as application-specific processors, general purpose processors, memory structures and communication networks. Therefore, the developer is faced with what Künzli, Thiele, and Zitzler call “a huge design space” [13]. The developer has to explore a solution in this huge design space that is functionally correct and that does not fail at anytime. In this paper we present an integrated approach for a model-driven software design in C that ensures real-time capability and functional correctness. As an example this approach begins with algorithm models of the system using Simulink block diagrams. The algorithm models are suitable to meet design decisions such as tuning the algorithm and partitioning between hardware and software. The models of the software portion are mapped on tasks and interrupt routines. Inchron’s real-time simulator chronSim [1] is used to define a task scheduling so that the system remains causal and does not violate real-time constraints. The simulator uses virtual prototype models of processors that add information about the timing of commands, the number of registers, and arithmetic units. This so-called real-time performance analysis in chronSim outputs information about lag time, and detects real-time violations even before the program runs on costly hardware. Lag time changes the algorithm behaviour and therefore might destabilise the whole system. In the integrated approach for a model-based real-time embedded software design the time lag values are back-annotated to the functional description in Simulink. There the algorithms can be verified and tuned further before the next iteration of mapping software models on tasks. This integrated approach is only possible by combining different model implementations at various abstraction levels.

After an overview about the related work we present the used design flow in Section 3. In Section 4 we focus on how to describe the sys-

*The work has been supported in part by the Energy Research Alliance of Lower Saxony.

tem using a model-based approach. After that we present the event-based simulation of chronSim and how to use it for system analysis in Sections 5 and 6. We conclude the paper with an outlook on future work.

2. Related work

For the design of tasks developers distinguish between dataflow-oriented, control-oriented, and protocol-oriented types. As an example, the embedded software of a communication system consists of protocol-oriented tasks for parsing and decoding or encoding data packets, dataflow-oriented tasks for compression and decompression, and control-oriented tasks to manage the user interface. Dataflow-oriented tasks ideally consume and produce data sequences and run continuously. Control-oriented tasks react to events from outside or to events created by other tasks. Their states can change erratically. Protocol-oriented tasks combine data-oriented and control-oriented features. They are parsing a data-flow-like bit stream and depending on the detected pattern their state changes to detect the next pattern. The three task types can be distinguished by data access and coding styles. Correspondingly there exist very specific model representations and design tools.

The Unified Modelling Language (UML) defines thirteen types of graphical representations, divided into three categories [2]. As an extension of UML, the Systems Modelling Language (SysML) is intended for general systems engineering applications [4]. As an example, Rhapsody and Statemate from Telelogic I-Logix Inc support UML and an early version of SysML. The constructs of UML allow the simulation of events in the system. Dataflow-oriented tasks are preferably drawn as block diagrams. Design tools like Synopsys' System Studio, CoWare's Signal Processing Designer SPW, and The MathWorks' MATLAB/Simulink support the developer with block diagram editors [5]. Protocol-oriented tasks are supporting the decoding or encoding of data packets in wireless communication systems and in bus networks. Hence design tools come with communication standard libraries. As an example, for the design of systems with bus networks in automotive applications, Vector Informatik offers a set of tools, where CANoe (CAN Open Environment) plays the role as simulation environment for a bus network with several electronic control units [6].

The previous design tools provide system level models, which have to be mapped onto task level models. This mapping procedure is supported by code generation which leads to a causal system of scheduled tasks. As an example, The MathWorks has released Simulink Real-Time Workshop and Simulink Real-Time Workshop Embedded Coder [5]. As Simulink is widely used for the functional design of signal processing and control systems, we explored a design flow using Real-Time Workshop code generation.

3. Tool flow

The contribution of this paper is a new model driven C-based design flow. In this section the used tool flow is explained which is shown in Figure 1 .

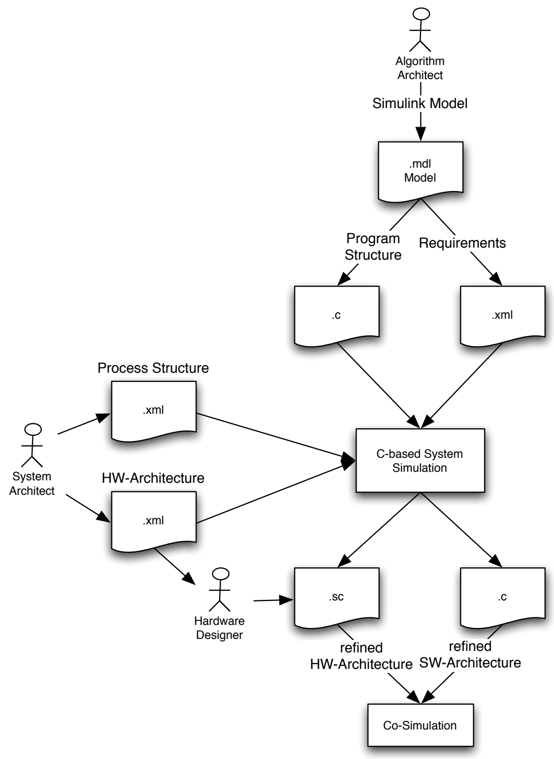

Figure 1. Platform design

The first step is the creation of the needed algorithms on system level. This is done by the Algorithm Architect as a block model in Simulink. Using the Real-Time Workshop and Real-Time

Workshop Embedded Coder it is possible to automatically generate the C-code for the software of the embedded system [5]. Additionally we enhance the blocks of the model in such a way that Simulink generates the (simulation-)requirements of the software during the simulation of the algorithms. These two factors define the input parameters for the C-based real-time system simulation in chronSim.

The System Architect can now define on which hardware (processor or microcontroller, bus systems, etc.) the previously generated software will be executed and divide the whole C-program into processes.

Now the real-time simulation in chronSim can be compiled and executed to get the information if the complete system is valid and can keep the deadlines [1]. Those informations are back annotated to the model and can be used to refine the software and the hardware to fit the needs. The Hardware Designer uses the specifications from the System Architect and the output from the simulation in chronSim and refines the Hardware in SystemC.

The resulting C-software and the SystemC-hardware code can then be executed and co-simulated.

In the following sections we want to describe the tools in detail and how to combine them for this model driven C-based system development.

4. Model-based system design

A model is a construct to describe the behaviour of the embedded system. Embedded systems consist of heterogeneous subcomponents that can be represented by different models. Models exist at different levels of abstraction that add structure to a design flow. With each level closer to hardware and software implementation more functional details are added. Models describing real-time constraints contain attributes like worst- and best-case response-times of tasks. As an example, Künzli, Thiele, and Zitzler [13] introduce five levels of abstraction on which design choices must be taken: (1) Logic Design and High Level Synthesis, (2) Programmable Architecture, (3) Software Compilation, (4) Task Level, and (5) Distributed Operation (system level).

Real-time design uses the fact that models at the task level describe the hardware and software portion with an interdependent relationship. As hardware affects task timing and scheduling, so-called virtual prototype models of the hardware are used in combination with task models. A hardware-software co-design of systems with this interdependency leads to a system model with different representations of hardware, software, and the environment.

In this paper we want to use Simulink to design the model for the algorithms. Simulink by The MathWorks is a tool for modeling, simulating and analysing multi-domain dynamic systems. Its primary interface is a graphical block diagramming tool and a customisable set of block libraries. It offers tight integration with the rest of the MATLAB environment and can both drive MATLAB or be scripted from it.

Together with virtual prototypes of microcontrollers and processors we then analyse the complete system using chronSim.

5. Event-based simulation

In real-time systems task scheduling depends on real-time constraints. The execution time from triggering a task to its response is constrained by a deadline. The response time of a task depends on the task complexity, the hardware, and interruption times by other tasks. The real-time behaviour of embedded software depends on the causality of tasks and their concurrent execution. Hence, exploring the real-time behaviour is tightly related to the scheduling algorithm of the Real-Time Operating System (RTOS) that determines how and when commonly shared resources are available for a specific task. A schedulability analysis answers the question if the task schedule does not violate causality and the execution times do not miss their deadlines.

5.1. Schedulability analysis

The schedulability analysis uses abstract representations of the tasks and analyses if tasks will finish their execution within predefined deadlines. The dataflow can be modelled using flow-graph models. The analysis of the interference of tasks on different priority levels will lead to a value of the worst-case response time (WCRT) of a task. Jersak, Richter, and Ernst propose a performance analysis as a promising alternative to simulation [12]. This analysis calculates minimum and maximum response times for tasks or task chains based on task properties, scheduling parameters and all possible timings of activating events. Therefore an event model, like the event stream model by Gresser [10], describes all possible timings of events. Such an analysis is based on the assumption that the worst-case execution time of each task is known [7][8].

There exist several design tools that support a worst-case execution time (WCET) analysis. As an example, AiT created by the company AbsInt is intended for this longest path analysis [11]. AiT computes upper time bounds for the WCET of the embedded system for all combinations of inputs and each task execution. Byhlin *et. al.* [9] explain that the analysis of the longest path requires additional information such as program flow, targets of indirect function calls and branches. According to Heckmann and Ferdinand [11] the upper bounds for the iteration numbers of all loops have to be known.

The longest path analysis is also suitable for non-preemptive tasks. It can provide very conservative estimates but the quality of results depends on constraints specified by the developer about software behaviour and hardware architecture. If the behaviour of the real-time system depends on data-dependencies a simulative approach using task models and virtual prototypes is an alternative to schedulability analysis.

5.2. Real-time simulation

A simulation of tasks on a virtual platform will lead to an average estimate of the WCRT because the models carry information about software and hardware. This simulation considers all unforeseen events and errors and takes them into account.

ChronSim is a platform for real-time simulation. With this real-time simulator the software runs on a host computer before task scheduling has been defined and before there exists any hardware. It comes with virtual prototype models of the target architecture that are adding information about the timing of instructions, number of registers and arithmetic units to the simulation. Additionally the simulation takes target compiler optimisation settings into account. Before execution of the simulation, the task source code is translated into an intermediate representation consisting of basic blocks. The runtime of these basic blocks is estimated using information about hardware allocation and linkage. This intermediate representation is converted into the machine language of the host platform afterwards. With this intermediate code, latencies and delays associated with the hardware add the notion of time to the running software.

The chronSim simulation combines analysis of the interaction and behaviour of tasks. The developer can chose virtual prototype models that are associated with an RTOS and its functions like OSEK [3] or generic prototype models in combination with chronSim-specific RTOS functions. Beyond that chronSim comes with generic functions that describe hardware behaviour such as processor allocation (blocking) and delay, protection of hardware resources using semaphore objects, and task synchronisation using event and queue objects. In the generic case, delays added in the intermediate representation approach are superseded. The developer can configure an entire artificial operating system by setting task priority, and combination of scheduling strategies such as “First in, First Out”, “Priority”, and “Round Robin” with scheduling types such as “Preemptive”, “Cooperative”, “Time-Slice”, and “Run-To-Completion”. The developer can chose between a functional and a structural representation of the embedded software. In the structural case, the developer models just skeletons of the tasks without any functionality. These skeletons are suitable to create a hypothesis of task scheduling and deadlines. In the further design flow, the developer can successively refine the model by adding functional code to the tasks. With each refinement step, the simulated real-time behaviour is approaching the behaviour of embedded software running on a virtual prototype model.

With the simulative approach the developer can explore the software architecture and task scheduling early in the design flow to check the systems behaviour if it fulfils its real-time constraints. Hence the integrated approach described here combines real-time simulation in chronSim with Simulink’s dataflow simulation.

6. Simulation-based analysis

The integrated approach which is presented here uses design representations for system-level and task-level analysis. At the system level the algorithm is modelled using block diagrams in Simulink. Simulink Real-Time Workshop generates C-code out of the block diagram. At the task level, the developer explores, at which rate the tasks should be activated and if all tasks do not exceed their deadlines. Figure 2 shows the integrated approach of system level analysis and task level analysis.

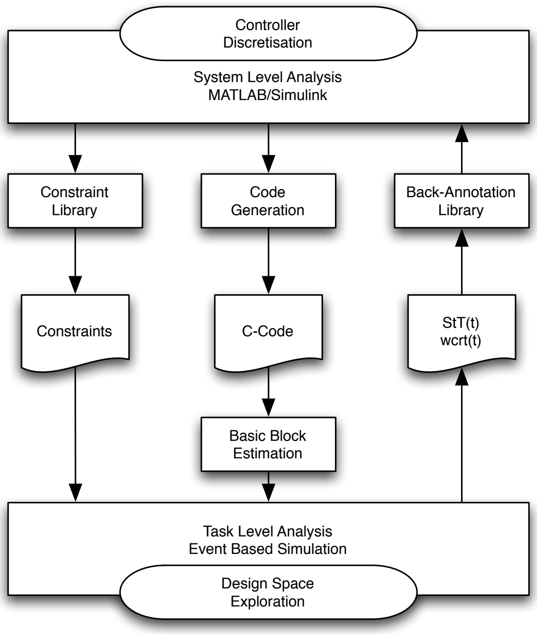

Figure 2. System level analysis and task level analysis with design files and annotations

The block diagrams in Simulink describe the overall system which consists of models of the digital controller, the plant controlled system, and the environment. In a first step, the developer explores controller model parameters to tune the overall system behaviour and stability. All models in these block diagrams are untimed and describe analog values. As tasks in embedded systems process quantised and sampled data and they add an execution time, models of the controller in the block diagram are replaced by a discrete representation. In the integrated approach, the controller models are simply switched to a different representation. This switch is performed by a dedicated constraint library provided to connect Simulink with chronSim and which is fully integrated in Simulink. New simulations in Simulink now show lag-times and quantisation levels. These simulations are suitable to detect the worst-case lag-times before the system reaches the stability limit. These lag-times define the deadlines that have to be watched at the task level. From the constraint library the worst-case response times, activation times, and quantisation levels are written to a constraints file. This file is available for task level analysis in chronSim. The developer uses Simulink Real-Time Workshop in parallel to generate source code for each task.

In chronSim the developer creates new tasks and adds the functionality of the generated source code. Additionally it is possible to combine the generated code with legacy or handwritten code of other system components. The constraints define the deadlines in the chronSim simulation. In case a deadline is missed, the developer has several options. Schäufele and Zurawka recommend that the developer might have to reconsider prolonging deadlines, change task scheduling by shifting activation times, changing task priorities, or splitting long running tasks into smaller chunks [14]. Especially prolonging deadlines and splitting long running tasks into smaller chunks requires a reiteration at the system level in Simulink.

After task level analysis and hardware/software design space exploration, the correctness of the algorithm has to be verified at the system level using simulations in Simulink once more again. The chronSim simulation outputs activation times and response times of tasks to a file which is back-annotated to the Simulink simulation. This file is read by Inchron's back-annotation library during a Simulink simulation where it affects the algorithm behaviour. At this point the algorithm simulation might show an unstable behaviour or simply output wrong results. After corrections at the system level in Simulink, like changes to the control parameters, the developer can start the next round in chronSim.

7. Case study: Submarine Explorer

The submarine exploring robot is a platform to support research on autonomous underwater vehicles. The mechanical system is based on standard components that are available for model construction of radio controlled submarines. The submarine explorer has been used to test the integrated approach for a model based software design.

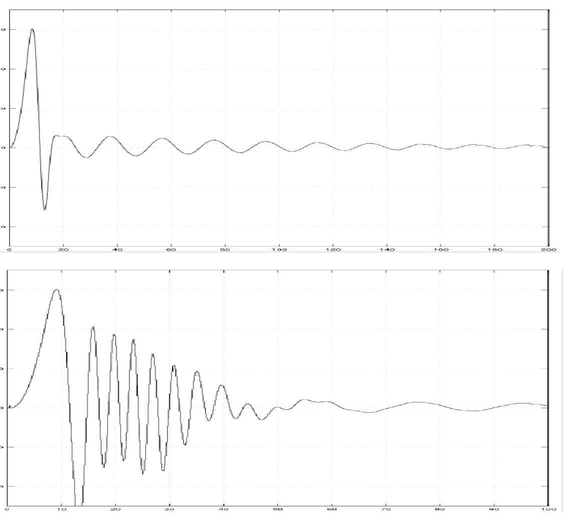

Figure 3. Output of course controller (continuous / with 200ms response time)

The initial algorithm simulation of the course control model in Simulink outputs a continuous response curve as shown in Figure 3 above. In case input data is sampled and the output is delayed with a time-lag, the system can become unstable. In the second example of Figure 3 the output signal shows an erratic behaviour. In this case the responding and sampling times are too long. The deadline for the task is shorter.

8. Conclusions

This paper presents an integrated approach for a model-based real-time embedded software design using C-code. As hardware affects task timing and scheduling, virtual prototype models of the hardware are used in combination with task models. For real-time behaviour exploration we give preference to a simulative approach instead of a static schedulability analysis because the developer can design deadlines and RTOS functions early in the design flow. The integrated approach combines real-time simulation in chronSim with Simulink system-level simulation. Delays and starting times of tasks are back annotated to the system-level where they are used to adjust the control algorithm parameters. This back-annotation of embedded software parameters to the system level supports a virtual prototype-based design flow that leads to a functionally correct and stable system.

References

[1] INCHRON GmbH. http://www.inchron.com, 2007.

[2] OMG's Unified Modeling Language (UML). http://www.omg.org, 2007.

[3] OSEK VDX Portal. http://www.osek-vdx.org, 2007.

[4] SysML Forum. http://www.sysmlforum.com, 2007.

[5] The Mathworks. http://www.mathworks.com, 2007.

[6] Vector Group. http://www.vector-group.net, 2007.

[7] K. Albers, F. Bodmann, and F. Slomka. Hierarchical Event Streams and Event Dependency Graphs: A New Computational Model for Embedded Real-Time Systems. In *IEEE Proceedings of the 18th Euromicro Conference on Real-Time Systems*, pages 97–106, 2006.

[8] F. Bodmann, K. Albers, and F. Slomka. Analyzing the Timing Characteristics of Task Activations. In *Poceedings of the SIES'06 (Symposium for Industrial Embedded Systems)*, 2006.

[9] S. Byhlin, A. Ermedahl, L. Gustafsson, and B. Lisper. Applying Static WCET Analysis to Automotive Communication Software. In *17th Euromicro Conference of Real-Time Systems (ECRTS'05), Mallorca, Spain*, 2005.

[10] K. Gresser. An event model for deadline verification of hard real-time systems. In *5th Euromicro Workshop on Real-Time Systems, Finland*, 1993.

[11] R. Heckmann and C. Ferdinand. Worst-Case Execution Time Prediction by Static Program Analysis. In *AbsInt Angewandte Informatik GmbH*, 2005.

[12] M. Jersak, K. Richter, and R. Ernst. Performance Analysis for Complex Embedded Applications. In *International Journal of Embedded Systems, Special Issue on Codesign for SoC*, 2004.

[13] S. Künzli, E. Thiele, and E. Zitzler. Multi-criteria Decision Making in Embedded System Design. In *SoC: Next Generation Electronics, IEE Press*, 2005.

[14] J. Schäufele and T. Zurawka. *Automotive Software Engeneering*. Vieweg, 2003.