Modelling and Verification of MPSoC

Aske Brekling

Kongens Lyngby 2006
IMM-M.Sc-2006-99

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Abstract

This thesis presents a formal model based on the system-level MPSoC simulation framework ARTS [10, 11]. This model formalizes all notions regarding timing and synchronization in such a manner that some properties of systems can be verified. The full semantics for the model is provided together with examples of how systems are specified in this semantics.

In developing Multiprocessor System-on-Chips (MPSoC), many interrelated choices have to be considered at the levels of the application, the operating system and the configuration of the platform. Choices regarding properties of systems have great consequences in unforseen areas of the system. This makes it a major challenge to develop correctly implemented MPSoC together with arguments for decisions leading to the solution.

Decisions leading to the implementation of such systems are many, and most are non-trivial and complex. In the development phase it is not enough to simply look at the different layers of the systems independently, as a minor change at one layer can greatly influence other layers. Tools providing means for analysis at system level (i.e. taking all layers into account) are in high demand. Especially tools for verification of properties of the systems are desirable, as verification can provide guarantees that the given criteria for the system properties are met.

Keywords: Timed automata, UPPAAL, Multiprocessor System-on-Chip (MPSoC), ARTS, Verification

Resumé

Denne afhandling præsenterer en formel model baseret på ARTS [10, 11], som er et "system-level MPSoC simulation framework". Denne model formaliserer alle begreber vedrørende timing og synkronisering på en sådan måde, at verifikation af visse egenskaber for systemer bliver mulige. Den komplette semantik for modellen fremlægges, kombineret med eksempler på hvordan systemer specificeres i denne semantik.

I Udvikling af Multiprocessor System-on-Chips (MPSoC) er det nødvendigt at overveje adskillige indbyrdes relaterede valg i alle lag af systemet; vedrørende applikationen, operativsystemet og i konfigurationen af platformen. Valg i forhold til systemegenskaber har vidtrækkende konsekvenser i uforudsete områder af systemet. Dette gør det til en større udfordring at udvikle korrekt implementerede systemer samt at give gode argumenter for beslutningerne førende til den endelige løsning.

Beslutninger førende til den endelige implementering af systemer af denne slags er mange, og de fleste er både ikke-trivielle og yderst komplekse. Det er ikke nok bare at se på hvert lag for sig i udviklingsfasen, eftersom en mindre ændring i et lag kan have stor inflydelse på andre. Der er derfor brug for værktøjer, der kan analysere på systemniveau, og således formår at tage alle lag i betragtning samtidig. Værktøjer der kan verificere systemers egenskaber af speciel nødvendig karakter, idet verifikation kan garantere, at de opstillede egenskabskriterier overholdes i de implementerede systemer.

Nøgleord: Tidsautomater, UPPAAL, Multiprocessor System-on-Chip (MPSoC), ARTS, Verifikation

Preface

This thesis was prepared at Informatics and Mathematical Modelling at the Technical University of Denmark in partial fulfillment of the requirements for acquiring a Master of Science in Engineering.

The thesis deals with aspects regarding formal modelling of intelligent embedded systems; in particular Multiprocessor System-on-Chips.

The project was completed in the period from April 18, 2006 to October 23, 2006 under the supervision of Associate Professor Michael R. Hansen and Professor Jan Madsen.

An abstract containing the major findings from this project was submitted and accepted for presentation at *Nordic Workshop on Programming Theory 2006 (NWPT06)*. The abstract was published in the proceedings for NWPT06, and the presentation was conducted at the workshop in October 2006 in Reykjavik, Iceland.

Aske Wiid Brekling

Lyngby, October 2006

Acknowledgements

I would first of all like to thank my supervisors Michael R. Hansen and Jan Madsen for their support and guidance. I have great appreciation for your support, motivation and the challenges you have given me through the period of time in which this thesis has been completed.

My gratitude also goes to Pavel Kozin, who has been a great support in the last half year, and for his willingness to let me borrow equipment which in part has made it possible to finalize the thesis.

I also must thank my family; my parents (Lisbeth and Anders), my sister (Iben) and my brother (Adam) - as well as their significant others - for their input and support on issues and challenges I have provided them with the past six months and especially the last couple of weeks.

Finally, a very special thanks to my wife Jeanifer Brekling for her endless support through the entire process of the project leading to this thesis. Also, I need to thank her greatly for her help and countless hours of proofreading.

Contents

Chapter 1

Introduction

In Design and development of Multiprocessor System-on-Chips (MPSoC) is both complex and expensive. Many non-trivial decisions must be made in order to create systems that are correct and efficient. This is in part due to the fact that anomalies are introduced when leaving the single processor domain and entering a multiprocessor domain. Also, the high degree of complexity of both hardware and software in MPSoC design makes design decisions non-trivial. This Master's thesis provides an overview of some choices that must be made in the design process, and a formal model for analysis of systems considering these choices has been developed and will be presented in detail. With this model in hand, a formal explanation and description of a MPSoC is available and certain types of verification are possible.

The purpose of this chapter is to:

- Provide an explanation of the components in a MPSoC
- Highlight some important choices that must be made in the design process
- Give insight into different ways to describe and model MPSoC
- Explain in which way timed automata can be used to model MPSoC
- Provide an overview of the thesis structure

Having read this chapter, the reader will have been introduced to the different concepts regarding MPSoC and the issues arising in the design process. Furthermore, the need for making a formal model for these systems will be understood. The reader will have obtained the necessary knowledge and understanding in order to relate to the following chapters.

1.1 MPSoC

A MPSoC is a full system of both hardware and software on a single chip. The hardware (or platform) is made up of multiple processing elements on which the embedded software (the application) is running. Today, systems like these are becoming more and more complex in design, and that requires a much more systematic approach to decisions made in the design process.

In the design process of a MPSoC many choices have to be considered; these include but are not limited to the following:

Number of processing elements Making decisions on how many processing elements are needed for the MPSoC to have the desired behavior is not a simple task. Introducing *additional* dedicated or programmable processing elements may provide extra processing power to the overall system, but this is not necessarily always the case. Furthermore, the *removal* of processing elements may make the overall system less efficient or powerful; however, this also is not necessarily so.

Type and structure of processing elements Processing elements on a MPSoC can be either dedicated to a specific task or operation or programmable and able to execute a wide range of different tasks. Programmable processing elements provide much more flexibility and efficiency to the overall system; however, they also require administration of the execution time and any dependencies the system might have. In order to make dynamic decisions on which tasks should be granted processing time on the processing elements, a need for real-time operating-systems (RTOS) arises. A RTOS manages the execution time on a processing element by scheduling the tasks to be executed and administers how dependencies among tasks are maintained and resolved.

Mapping the application onto the platform Choosing which tasks should be run on which processing elements is not simple either. Moving the execution of one task onto a different processing element could benefit the overall system in terms of faster execution or less preemption and thereby

context switching; however, the result could also be the exact opposite, with slower overall execution and more preemption. A good mapping of the embedded application onto the chosen platform therefore becomes a complex task when creating a good MPSoC in terms of correctness (making timely deadlines), efficiency and cost.

Choice of operating system As mentioned earlier, a need for RTOS is introduced when several tasks are mapped onto the same processing element. This choice greatly affects the behavior of the MPSoC and should therefore be considered carefully. Using inspiration from the ARTS [10, 11] system, three issues that describe the behavior of an operating system are identified: synchronization, allocation and scheduling.

Synchronization Different tasks of the application might share variables, and execution of certain tasks might be required before others can run. Causal dependencies like these require a mechanism to administer the information regarding dependencies and keep track of which dependencies are resolved and which are not. This mechanism is the synchronizer. Each time a task finishes execution, it checks whether dependencies have been resolved and resets the dependencies when needed.

Allocation Different tasks might require access to the same resources. These could be communication channels, memories, etc. This access is administered by a mechanism that, much like described for the synchronizer, keeps track of and grants the different tasks access to shared resources.

Scheduling In order to make a dynamic decision of which task is the most suitable for execution on a given processing element, a scheduling principle must be in place. Scheduling principles can either be static or dynamic. A static principle uses a set criterion. This could either be user defined or based on a characteristic of the given task, e.g. its time period. A dynamic principle uses dynamically updated criteria for making scheduling decisions, e.g. the next deadline of a task. It is clear that a dynamic scheduling principle can make a more informed scheduling decision; however, dynamic scheduling also requires more administration than static.

Due to all the freedom in choices at the various levels of abstraction that must be taken into account, it is a major challenge to develop a correctly implemented MPSoC together with good arguments for the choices leading to the solution.

1.2 Related Work

Related work in the area of modelling MPSoC in connection with analyses in different design phases can be roughly divided up into simulation-based approaches and more formal approaches.

In simulation-based approaches, SystemC is a de-facto language for system modelling [9, 12]. Hessel et al. [7] and Moigne, Pasquier and Calvez [12] propose system-level MPSoC models for analysis in design space exploration of real-time systems. Development of models for RTOS provides the designer with a quick evaluation of different scheduling and synchronization mechanisms, according to the properties of the RTOS, in order to validate the dynamic real-time behavior of the system. In [12], the possibility of verification through simulation is proposed as future work. Loghi et al. [9] and Fummi et al. [6] provide SystemC-based models for exploration of different types of communication in MPSoC (i.e. different network configurations). MPARM [9] is one of the most advanced multiprocessor cycle-accurate architectural simulators based on SystemC. In [13], the use of SystemC in transaction-level modelling (TLM) is discussed. This approach is primarily motivated by the ease of describing MPSoC models for on-chip busses at different abstraction levels. ARTS [10, 11] is a SystemC-based simulation framework for MPSoC. In ARTS, the designer can, at various states in the process, make simulations analyzing system properties such as timing, memory and power usage and communications.

Although simulation provides valuable input and feedback in the design process, a need for verification and formal models describing the MPSoC arises. Formal models can offer systematic verification and provide guarantees for properties and bounds for critical performance parameters.

In order to handle shortcomings of simulation-based approaches, several more or less formal approaches have been proposed. In [14], an approach for analyzing communication delays in message scheduling, together with optimization strategies for bus access, is presented. Thiele et al. [17] provides a real-time calculus for scheduling of preemptive tasks with static priorities. Richter, Jersak and Ernst [15] propose a formal approach based on event flow interfacing. A simple tool for analysis interfacing is provided, together with analysis algorithms, for configuring a global analysis model.

Although different formal approaches for modelling MPSoC have been given here, none of these capture the mathematical meaning of all concepts involved. At the moment, there is no approach (to my knowledge), with which system-level MPSoC - where all notions are captured in a formal manner - can be defined such that the mathematical meaning for all components is fully specified.

1.3 Timed automata for MPSoC modelling

Many of the decisions made in the design of MPSoC are in some way connected to either timing or synchronization: schedulability, communication between different components, communication between different layers (application, RTOS and platform), etc. With these issues in mind, using timed automata [1] to model such systems seems an obvious choice. A timed-automata model can formalize these aspects, and the model can be subject to verification through model checking using a tool like UPPAAL [3, 8]. In the following, the reader is assumed to have general knowledge of timed automata; if not, appendix A provides an introduction.

The model provided in this thesis is not an exact model as time is modelled discretely - more on this in section 3.3.4. This discretization is done in order to deal with preemption. In [5], a strategy for determining schedulability for preemptive scheduling strategies is proposed. This strategy is used in TIMES tool [2] when determining schedulability; however, certain restrictions apply. First, only the single processor domain can be examined; and second, combining dynamic scheduling and dependencies is not possible. It should be noted that in single processor cases with either dynamic scheduling or dependencies, TIMES tool provides very useful results for schedulability analysis.

To make a model without descretization of time, stop-watch automata would be an option. However, it is known that the reachability problem for stop-watch automata is undecidable [5]. An older version of UPPAAL, where stop-watches are implemented, exists. This could be used for simulation, but in personal communication with Kim G. Larsen from Aalborg University, it was explained that verification in this system is conducted using over-approximation, which means that a deadlock-free system can be guaranteed deadlock free through verification while detection of a deadlock in a system could simply be due to the over-approximation.

1.4 System-Level MPSoC Framework

The following description of system-level MPSoC framework is a generic description. However, the terminology is inspired by that of the ARTS system [10, 11].

There is a trend that modern hardware systems are moving toward *platforms* made up of multiple programmable and dedicated *processing elements* (*PE*)

implemented on a single chip; in other words MPSoC. On these processing elements, different parts of an embedded *application* are running. The application can be divided up into a number of *tasks*, each of which represents a piece of sequential code. These tasks might have timing constraints as well as causal dependencies with other tasks (e.g. certain tasks will have to finish before others can run).

Having processing elements that can execute several different tasks and manage execution time dynamically introduces a need for a dedicated *real-time operating system* (*RTOS*) as a layer between the application and the hardware platform. The RTOS should manage scheduling of the different tasks as well as the dependencies tasks might have with each other.

A *system-level framework* is a cross-layer model and should be recognized as a framework of the overall system comprising application, RTOS and processing elements. The system-level view of a system becomes important, as a change one place in the system may have effects in other unforseen places. For example, a minor change in operating system behavior may have a great impact on how the application executes. There are currently tools available for simulation of MPSoC; one of these is ARTS [10, 11]. Figure 1.1 provides an overview of the different components of a system-level framework for a MPSoC. For a more detailed explanation on this, see chapter 2.

1.5 Problem Formulation

With this background, the problem at hand can be formulated. How can a formal model be developed using ARTS as a basis (with its modular structure and notions defining MPSoC)? The model should support simulation and verification on a formal basis and assist in analysis of correctness and performance properties within the design phase of MPSoC.

1.6 Thesis Structure

The thesis is structured as follows:

- Chapter 2 gives general understanding of the ARTS system that is used as a basis for the development of the formal model.

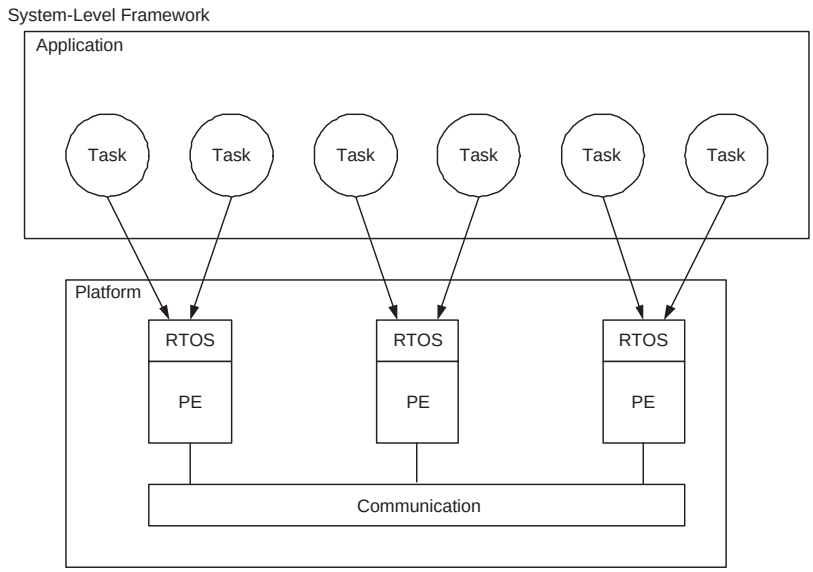


Figure 1.1: System-Level Framework for a MPSoC

- Chapter 3 provides the formal model, with timed automata for each component in the modular structure and explanations for non-trivial development steps taken.
- Chapter 4 explains how UPPAAL can be used for simulation and verification in connection with the models developed.
- Chapter 5 provides a Java frontend to create the input to the UPPAAL system in a structured and simple way, and an example of usage is given.
- Chapter 6 gives MPSoC examples, and results of verification of the UPPAAL models are described and analyzed.
- Chapter 7 discusses areas for further development and provides suggestions for how this can be done.
- Appendix A offers an introduction to timed automata. The source code for the Java frontend implemented is given in appendix B. The Java source code for the examples explained in chapter 6 can be found in appendices C and D. A full timed-automata model for the example given in chapter 5 is provided in appendix E. Finally, a description of the content of the attached cd can be found in appendix F

Chapter 2

The system-level framework ARTS

In this chapter the different components of the ARTS [10, 11] framework are explained in detail, as well as how some features (e.g. that of the modular structure of ARTS) provide a good basis for developing a model that formally explains and describes ARTS and MPSoC in general. Furthermore, it will be highlighted how the model to some extent is explicit, but also how some parts are only informally explained except through the implementation of the simulation and the behavior of the simulation itself. This provides evidence as to why a formal model is needed.

2.1 Overview of ARTS

The framework ARTS models a MPSoC at system level. It can be explained and understood through Figure 1.1. In short, an ARTS model of a MPSoC is a mapping of an application onto a platform. Once the application, the platform and this mapping are specified, ARTS provides a simulation engine where different properties of the MPSoC can be examined.

The full ARTS model has a modular structure. This means that each component

is a stand-alone submodel. This feature makes the model flexible, as it is easy to interchange components, but it also provides a good basis for developing a formalized model as the formal model can preserve the modularity and specify the full system in terms of smaller subsystems. Through this modularity some of the complexity in understanding the full system is removed.

2.2 Application

An application in ARTS is modelled as a number of tasks. Basically, the overall application is divided up into several pieces, each of these representing a piece of sequential code. These tasks might have dependencies among each other (due to shared variables, synchronous communication, etc.) so that certain tasks must finish execution before others can start.

2.2.1 ARTS Task Model

A task in ARTS is an abstraction of a sequential process of the application capturing timing, synchronization and causal dependencies with other tasks. Although ARTS provides support for non-periodic tasks, here - for simplicity - a task is assumed to be periodic and to require a certain execution time in each period (i.e. it does not miss any deadlines). Its timing constraint is that it achieves the required execution time in each period. The model of a task in ARTS can be seen in Figure 2.1 as a finite automaton decorated with real-valued variables to model the timing. It is based on the model introduced in [10].

The timing constraints of the task are represented by the real-valued variables (clocks) `cp` and `cr`. `cp` is the clock describing the periodity; it is reset to zero when a new period starts. `cr` is the clock describing the time remaining for the task to complete the execution in the current period. It is reset to the execution time at the start of each period. It should be noted that in ARTS notions of offset and deadline are also included. However, these are only implemented in the simulation framework and not explicitly explained. In section 3.1, a more in-depth explanation of the notions of offset and deadline can be found.

In state *Idle* the task has finished the execution of the current period and is waiting for the next. In the *Ready* state, the task has started a new period and is waiting for a `run` signal from the platform to start execution. *Running* is the state where the task is executing. While the task is in this state, `cr` decreases.

When the task is in the *Preempted* state, execution of it is temporarily halted so that another task (with higher priority) can be executed. The task waits for a `run` signal from the platform to resume execution.

2.3 Platform

A platform in ARTS is made up of a number of processing elements, each of which runs its own (potentially different) RTOS. In Figure 2.2 the structure of one processing element in ARTS is given. In the following, each component (i.e. synchronizer, allocator and scheduler) of a processing element will be explained in detail.

2.3.1 Synchronizer

One of the basic services provided by the RTOS is synchronization among tasks running in parallel. This enables ARTS to model real-time systems with data dependencies. If, for example, the task τ_2 needs data computed by τ_1, it will have to wait for τ_1 to finish before it can start executing. These tasks can be mapped to the same or to different processing elements. In short, the synchronizer manages causal intra- and interprocessor dependencies among tasks.

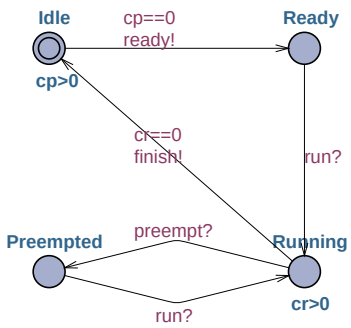


Figure 2.1: Model of a task in ARTS [10]

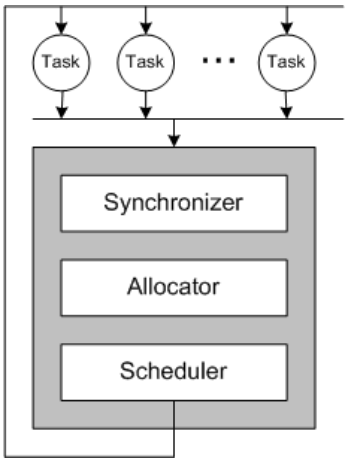


Figure 2.2: Structure of a processing element in ARTS [10]

2.3.2 Allocator

In a real-time embedded system tasks often share resources. This could, for example, be several concurrent tasks competing for utilization of shared memory. The allocator provides resource allocation to the system.

2.3.3 Scheduler

The major job of a RTOS from a system level is task scheduling. The scheduler dynamically chooses which task should be executed on the processing element from the set of tasks deemed ready for execution. This choice is made according to the scheduling policy of the scheduler. Scheduling policies are either static or dynamic. Differences between static and dynamic scheduling principles are explained in section 1.1. The following will provide explanations of four different scheduling principles:

User defined fixed priority scheduling (FP) In FP scheduling, each task is given a set priority. The task with the highest priority is selected by the scheduler. FP is a static scheduling principle.

Rate monotonic scheduling (RM) Scheduling using RM bases the scheduling decision on the period of the task. The task with the shortest period

has the highest priority and is therefore selected. RM is also a static scheduling principle.

Deadline monotonic scheduling (DM) In DM scheduling the deadline is used instead of the period, i.e. the task's static deadline value. This means that the task with the shortest deadline has the highest priority and is selected by the scheduler. DM is also a static scheduling principle.

Earliest deadline first scheduling (EDF) Scheduling with EDF uses the time remaining until the next deadline for the task as criterion for scheduling. The task with the first arriving deadline has the highest priority and is selected. EDF is a dynamic scheduling principle as deadline information is maintained dynamically.

Note that in ARTS only RM and EDF are implemented; however, the framework structure makes it easy for the designer to implement new modules for other principles as needed.

The job of the scheduler is simply to issue the `run` and `preempt` signals received by the task as seen in Figure 2.1.

2.4 Discussion

As mentioned, ARTS provides a simulation engine given a platform, an application and a mapping of the application onto the platform. Although simulation can provide some idea of how the system works, it does not guarantee the correctness of the system or give any other guarantees. Furthermore, the level of understanding of the model only reaches as far as the simulations themselves. A more detailed overview and broader understanding of the MPSoC is not obtained, as some parts are only defined informally or as SystemC code. A full formal model describing the components of ARTS, using the modular structure of ARTS as a basis, is therefore a desirable solution.

Chapter 3

Timed-Automata Semantics

In this chapter, a semantics for an important subset of ARTS - in the form of timed automata [1] - is provided. First, the scope of the subset of ARTS, included in the semantics, is explained. The structure of the semantics is then given, together with detailed explanations for each component. This semantics opens up for a more explicit understanding of ARTS systems. Representing the semantics in the UPPAAL model [3, 8] further enables mechanisms for verification through model checking (more about this in section 4.2). It is assumed that the reader has general knowledge of timed automata; if not, appendix A provides an introduction to timed automata.

3.1 The subset of ARTS examined

The semantics provided contains formal models for an important subset of ARTS. We consider tasks to be cyclic and preemptive and have static values for period, deadline, offset and execution time. Furthermore, all processing elements perform synchronization and scheduling in zero time. These limitations have been introduced in order to simplify the problem at hand, but the semantics provides a basis for further development in which some of these limitations could be removed and the included subset of ARTS expanded.

3.1.1 Offset and deadline explained

Notions of offset and deadline are implemented in ARTS, but not explicitly explained. Therefore, the following will serve as an informal explanation of the concepts, and their formal meaning can be deduced from the timed-automata model:

Offset Since all tasks' periods are not necessarily aligned, i.e. not all tasks start at the same time, the concept of an offset is introduced. An offset is the number of time units later than the global start of the full system the task is released starting its first period.

Deadline A deadline is the number of time units the tasks has to finish its execution. In many cases deadline is modelled to be the same as its period; however, one may wish to model a task having a deadline different from the period. In the current model it is assumed, that the deadline does not exceed the period, i.e. $\forall task \in Tasks : deadline(task) \leq period(task)$ where $Tasks$ include all tasks in the system and *deadline* and *period* extract integers representing these for the given task.

3.2 Structure of the Semantics for ARTS

Each ARTS system is expressed as a collection of timed-automata, combined in parallel and communicating with each other using shared variables and synchronous communication via channels.

The structure of the semantics is described as follows:

$$
\begin{aligned}
&System = Application \parallel Platform \\
&Application = \parallel_{i=1}^{n} Task_i \\
&Platform = \parallel_{i=1}^{m} PE_i \\
&PE_i = Controller_i \parallel Synchronizer_i \parallel Scheduler_i
\end{aligned}
$$

where $\parallel$ denotes parallel composition of timed automata.

Notice that the basic structure of this semantics follows the description of ARTS in Figures 1.1 and 2.2, except that the semantics does not have an explicit component called an allocator and a controller component is introduced in the semantics.

3.2.1 Allocation

The purpose of the allocator in ARTS is to synchronize access to shared resources, such as communication channels, memories, etc., on the platform. These shared resources are, in our semantics, considered special tasks running on special processing elements, and the allocation mechanism can be expressed in terms of synchronization of these special tasks. For example, a shared bus would be represented by a special processing element and the use of the bus as a special task on this processing element. It should be noted that this representation does not fully capture the notion of resource allocation, due to the fact that only periodic tasks are implemented. Resource allocation represented in the manner described here should be based on non-periodic triggered tasks. Further development could provide an explicit model for resource allocation, more on this in chapter 7.

3.2.2 Communication between Platform and Application

A controller is introduced to govern the overall communication between the application and the platform. The controller is distributed to the individual processing elements. This component was introduced to be able to keep the original ARTS modularity, such that the synchronizer addresses synchronization issues only, and the scheduler addresses scheduling issues only. Introduction of such a component in ARTS could help explain and preserve the overall modular structure.

3.3 Application/Tasks

To give the semantics of the application, we must provide a timed automaton for each task. The aim is for the overall structure of a task from Figure 2.1 to be visible in the corresponding timed automaton.

However, to give the semantics, we must be explicit with regard to the timing and scheduling details of tasks. For example, in the ARTS model in Figure 2.1, it is implicit that `cp` is reset to 0 at an appropriate time when the task is in the *Idle* state. Such implicit assumptions must be made explicit in the semantics.

In the ARTS model it is possible to have offset and deadline properties of periodic tasks. This is also included in the semantics, see section 3.1.1.

The semantics should also allow for different scheduling disciplines. In this case we support earliest deadline first, rate-monotonic scheduling, deadline monotonic scheduling and fixed priority scheduling, these are explained in section 2.3.3. This, of course, complicates the semantics.

In order to provide a better understanding, the task was divided up into five different automata, one for the overall control and one for each of the four original states in ARTS from Figure 2.1 - *Idle*, *Ready*, *Running* and *Preempted*. The structure of the task can be explained in terms of the following parallel composition of timed automata.

$$Task_i = TaskControl_i \parallel Idle_i \parallel Ready_i \parallel Running_i \parallel Preempted_i$$

The following sections explain these five timed automata in detail.

3.3.1 Task Control

This component was created to provide an overview and better understanding of how the task works. It also keeps the same structure as the task model from ARTS. In short, the automaton contains four states and six transitions. Each of these transitions are divided up into three transitions between committed states in order to transfer control from the underlying automata representing each of the original states from the ARTS task model. The timed automaton for the task control can be seen in Figure 3.1.

Tasks Finishing Execution When a New Period Starts

In reality, the only difference between this component and the original task model is that one extra transition was added between the *Running* and the *Idle* state. This addition was made in order to be able to handle the situation where a task finishes its execution of one period in the same time unit as the next period starts. In the original implementation of ARTS this special case was not handled, so the formal model provides feedback for the original model as a potential area of improvement.

Comments to the Semantics

More detailed comments to the semantics of the task control are appropriate.

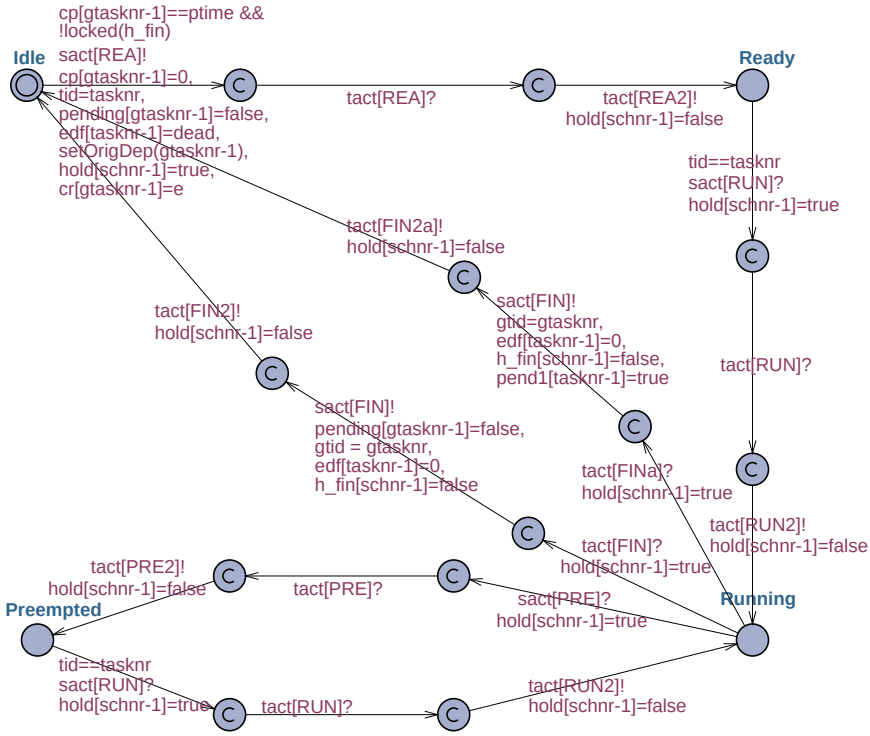


Figure 3.1: Semantics for the task control

Communication on channel arrays Two different arrays of channels are used to provide communication between the different timed automata. The array `tact` provides communication between the different components of the task, and the array `sact` provides communication between the application and the platform.

Communication between task components Each task has its own array of channels for communication with the other task components. Each index in these arrays provides communication between two specific components of the given task. For example, the channel `tact[REA]` handles the communication from the *Idle* to the *TaskControl* automaton and `tact[REA2]` handles communication from *TaskControl* to *Ready* automata.

Communication between platform and application Each processing element has an array of channels on which it communicates with the tasks mapped to it. Each index in these arrays corresponds to a signal in the ARTS task model in Figure 2.1: The channel `sact[REA]` corresponds to `ready`, `sact[RUN]` to `run`, `sact[PRE]` to `preempt` and `sact[FIN]` corresponds to `finish`. To identify the specific task that the signal in question belongs to, two shared variables `tid` (local task identification) and `gtid` (global task identification) are used.

For `sact[REA]`, `tid` is set to the local task identification `tasknr`; this tells the processing element which task is ready (basically a synchronization with value transfer).

For `sact[RUN]`, the platform sets `tid`; this specifies which of the ready tasks should react to the signal, and the transition has a guard limiting only the specified task to synchronize with the `sact[RUN]` signal.

For `sact[FIN]`, `gtid` is set to the global task identification `gtasknr`, telling the processing element which task has finished. This is again synchronization with value transfer. `gtid` is used, due to the fact that dependencies can be inter-processor dependencies. And dependencies are resolved through `sact[FIN]` signals.

Considering that only one task can execute on a given processing element at a time, `sact[PRE]` is issued without further identification.

3 transitions in 1 In order to control the flow of action through the task model and to keep each of the three transitions corresponding to one transition in the ARTS task model, the boolean array `hold` was introduced. This ensures that when a transition from one of the original states has begun, it continues without interruption until it reaches another of the original states. The main reason for making three transitions is that in UPPAAL, it is only possible to synchronize on one channel for each transition.

Local variables global Since the task model is now five different automata, locally accessible variables and clocks are needed to be known by different components; this means that `cp` and `cr` are represented in a global array with the index representing which task is concerned.

A cyclic task and its period In the task model from ARTS it is implicit that in each period the clock `cp` is reset. In the semantics this is now explicit. However, instead of resetting this clock externally, the transition is no longer taken with the guard `cp=0` but instead the guard `cp=ptime` (where `ptime` is the period of the task). With this transition, the update `cp=0` is completed and the cyclic behavior is modelled.

Locking mechanisms In ARTS, no formal explanation is given for how the actual flow of action on a processing element is performed. This is specified in the semantics through a range of locking mechanisms. `pending` ensures that the next time unit of a task running will not start until all `sact[REA]` and `sact[FIN]` signals have been reacted on. `pend1` ensures that the processing element reacts on all `sact[REA]` signals available in the given time unit. `h_fin` ensures that the processing element will react on a `sact[FIN]` signal (if one is available) before reacting on `sact[REA]` signals.

Dynamic scheduling and maintaining dependency information Allowing dynamic scheduling and dependencies requires some administration. A task starting its period must reset its dynamic criterion and the dependency information. In the semantics, EDF scheduling is the only dynamic scheduling principle. The array `edf` contains the deadline information for all tasks on the processing element. The function `setOrigDep(t)` resets the dependency information for the given task `t`.

3.3.2 Idle

When a task is located in the *Idle* state of the task model in ARTS, it basically waits for the next period to start. However, in the semantics some administration must be made. At the start of a system, criteria for static scheduling principles are set, and if the task has an offset, this must be observed and dealt with. Also, as the semantics prioritizes `sact[FIN]` signals over `sact[REA]` signals, these are announced in advance. Figure 3.2 contains the timed automaton representing the *Idle* state.

The states *Start* and *Offset* provide the administration regarding the start of the system. When leaving these two states, it will never return. Criteria for static scheduling are set using the arrays `pri` (for user defined scheduling), `per`

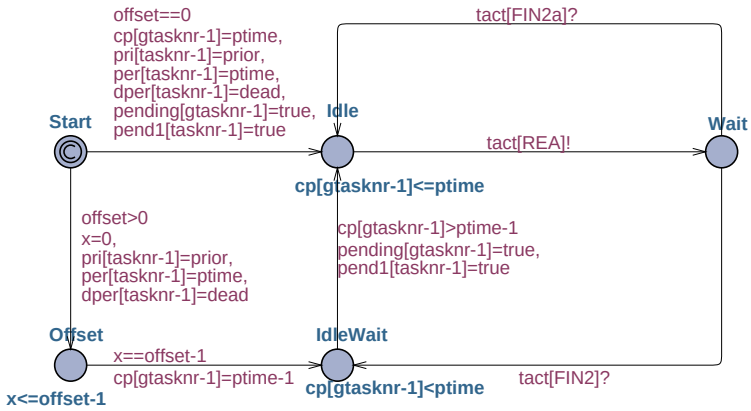


Figure 3.2: Semantics for Idle

(for rate monotonic scheduling) and `dper` (for deadline monotonic scheduling). If the task has an offset, the local clock `x` is used to keep the task in the *Offset* state for one time unit less than the duration of the offset after which it advances to *IdleWait*. The task will - if it does not have an offset - advance to the state *Idle* and, in doing so, set `cp` to the period of the task `ptime`. In this way, all tasks start at the end of a period. The state *IdleWait* is where a task waits after finishing execution and until the last time unit of its period. Then it continues to *Idle* and from here control is transferred to the *task control* automaton by advancing to the *Wait* state.

3.3.3 Ready

As was the case with *Idle*, no formal description for the *Ready* state of the task model is specified. However, with dynamic scheduling like EDF, the criteria must also be maintained in this state. Each time unit, the task will have the choice of receiving a `sact[RUN]` signal from the platform and moving to *Wait* (`sact[RUN]` is actually received by the *TaskControl* automaton, which then issues a tact[RUN] for the *Ready* automaton) or reducing its deadline in the array `edf` by moving to *Ready2*. The time units are maintained by the local clock `x`. To make sure that the task can receive a `sact[RUN]` signal from the platform, a locking mechanism in the form of the array `h_edf` ensures that all tasks are in the state *Ready1* when this signal can be issued. In Figure 3.3 the timed automaton for *Ready* is given.

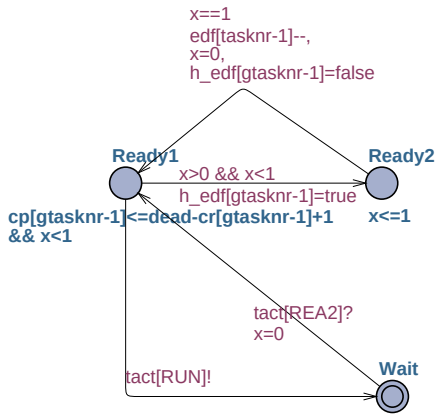


Figure 3.3: Semantics for Ready

3.3.4 Running

When in the *Running* state of the original task model in ARTS, the task is actually executing on the processing element. This means that the execution time left for the task must be decremented and the dynamic scheduling criteria must be updated (the deadline comes closer).

Here, a discretization of the running time of the task is introduced. As explained in section 1.3, this is done in order to deal with preemption and avoid stop-watches and over-approximation. Each time unit the task is in the *Running* automaton, it will make a round through the four states, *Running1*, *Running2*, *Running3* and *Running4*. In the transition from *Running3* to *Running4* the time unit is subtracted from the variable `cr` representing the remaining running time.

In order to cope with preemption and dependencies, it must be possible to preempt a task. To deal with timing issues, this has been modelled such that preemption can only happen at the top of a time unit (not during a time unit), but making sure that no preemption is done in the middle of a time unit requires some locking mechanisms. In Figure 3.4 the timed automaton for *Running* is given.

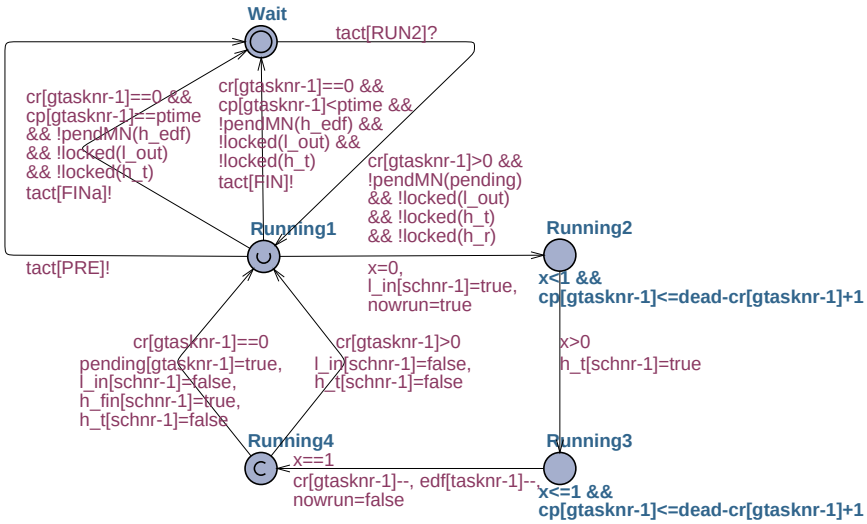


Figure 3.4: Semantics for Running

Maintaining execution time and deadline information

The main purpose of the *Running* automaton is to model that the task is executing on the processing element. This means that the remaining execution time (modelled by `cr`) must be decremented each time unit the task is executed on the processing element. The deadline information needs to be maintained as well, just as described in *Ready*.

Synchronizing tasks to allow for preemption

Locking mechanisms are used in order to maintain control. To make sure that all tasks currently running on all different processing elements are synchronized, `h_t` ensures that all tasks meet in the urgent state *Running1* before making the next decision - i.e. starting the next time unit - much like the *barrier* concept used in parallel computation. This is needed in order to be able to preempt a task (T_1) if a task (T_2) on another processor finishes and this resolves a dependency on the processor on which (T_1) is executing.

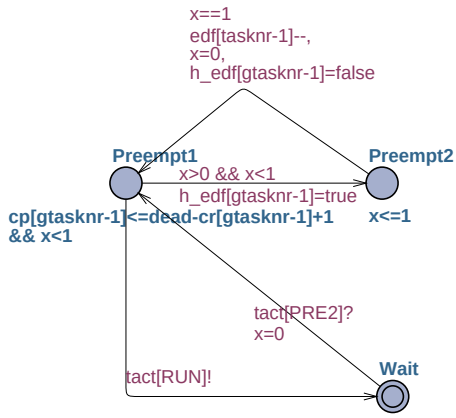


Figure 3.5: Semantics for Preempted

Locking mechanisms for reschedule

When a task finishes it might resolve a dependency. If this is the case, a reschedule (`resch`) signal is broadcast to all other processing elements. This prompts them to check whether `sact[PRE]` and/or `sact[RUN]` are needed in order to uphold the given scheduling principle. To make sure that all currently running tasks are able to react to a preempt signal originating from a reschedule, `l_out` ensures reschedule signals are reacted on before the next time unit is begun. `l_in` ensures that the processor on which the rescheduling originates finishes its local scheduling before other processing elements react to the rescheduling.

3.3.5 Preempted

As was the case with *Ready*, when having dynamic scheduling like EDF, the criteria must also be maintained in this state. Each time unit, the task will have the choice of receiving a `sact[RUN]` signal from the platform (`sact[RUN]` is actually received by the *TaskControl* automaton which then issues a `tact[RUN]` to the *Preempted* automaton) and moving to *Wait* or reducing its deadline in the array `edf` by moving to *Preempt2*. The time units are maintained by the local clock `x`. To make sure that the task can receive a `sact[RUN]` signal from the platform, a locking mechanism in the form of the array `h_edf` ensures that all tasks are in the state *Preempt1* when this signal can be issued. In Figure 3.5 the timed automaton for *Preempted* is given.

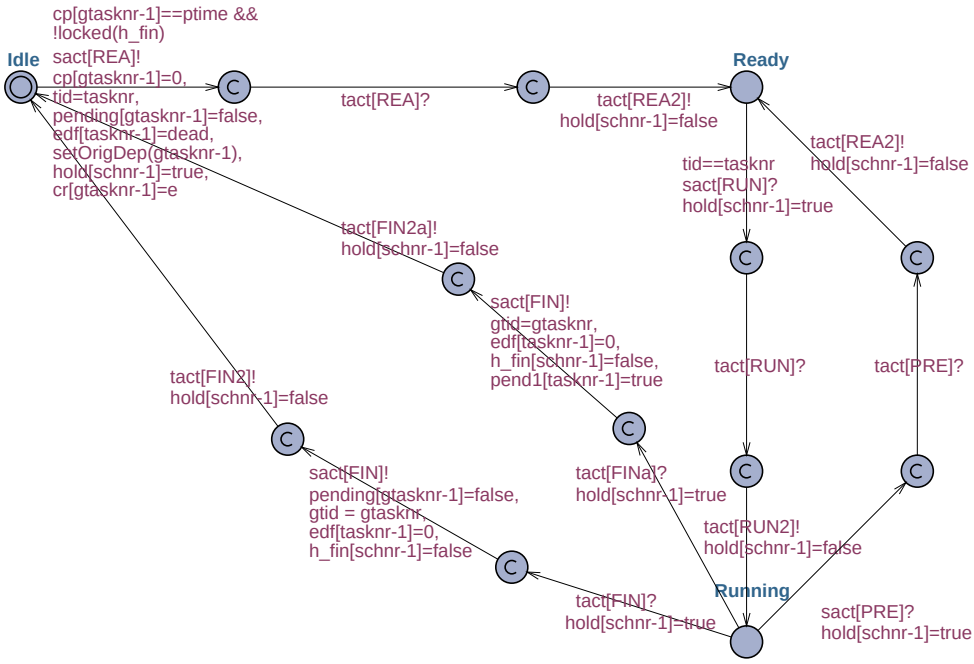


Figure 3.6: Semantics for modified Task Control

3.3.6 Comments to the application model

It is clear that the timed automaton provided for *Ready* and the one given for *Preempted* in reality are the same. The question then is whether the model for a task should not be simplified and the *Preempted* state be removed, since a task that is preempted is really just like one that has just become ready. The timed automaton for the *Task Control* is modified to model this behavior and can be seen in Figure 3.6. This simplification does not change the way the system is modelled, but rather simplifies it in regard to conceptual understanding and complexity of the model.

3.4 Platform

To give a semantics to the platform, timed automata for each controller, synchronizer and scheduler in the system must be provided. Since there is no formal model for any of these, the semantics will formally model how each of these components in ARTS works.

3.4.1 Controller

The controller receives signals from the tasks. Any `sact[REA]` or `sact[FIN]` signals issued by the tasks on that processing element are collected by the controller. The array `rbtid` contains information on the tasks that issued `sact[REA]` signals, and `ftid` contains information about the finishing task (if any task has finished). It then directs the control to the synchronizer through the channel `cact[SYN]`, and when the synchronization has been completed it receives a signal back through `cact[SYN]` and then directs control to the scheduler through the channel `cact[SCH]`. After completion of the scheduling, it receives a signal back through `cact[SCH]` and the controller once again waits to collect signals from the tasks. In the case where a dependency in another processing element is resolved, the controller (besides receiving `sact[REA]` and `sact[FIN]` can also receive `resch`. In receiving `resch` the controller waits for the processing element which issued the `resch` to finish its synchronization and scheduling. It then starts its own scheduler in order to check whether the resolved dependency should change which task is executing on this processing element. The timed automaton for a controller can be seen in Figure 3.7.

3.4.2 Synchronizer

The synchronizer maintains which tasks are ready to be scheduled based on whether or not they have unresolved dependencies. The state of these tasks' readiness can either be changed by a single `sact[FIN]` signal or a number of `sact[REA]` signals. In Figure 3.8 the timed automaton for a synchronizer is given.

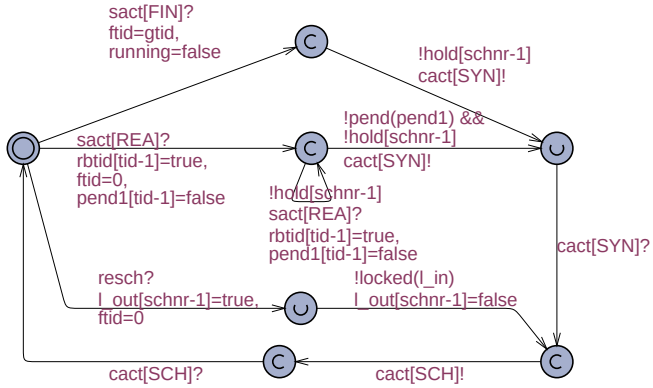


Figure 3.7: Semantics for the Controller

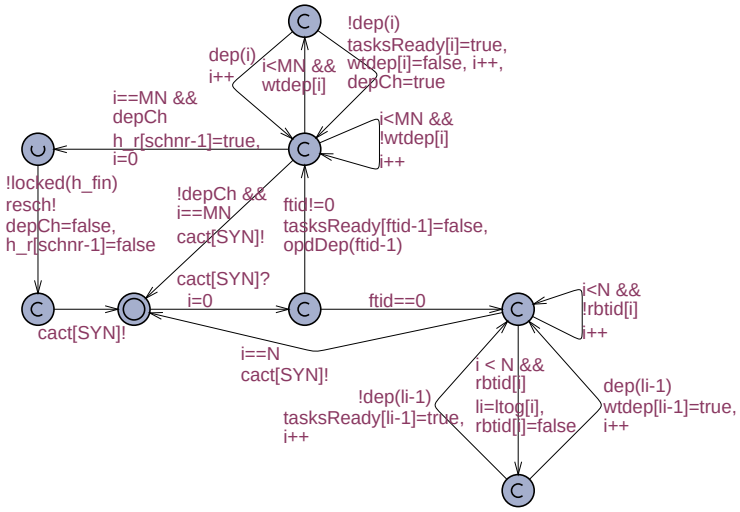


Figure 3.8: Semantics for the Synchronizer

A number of sact[REA] signals

If the synchronizer is reacting to a number of `sact[REA]` signals, `ftid` will be set to 0 and the transition guarded by `ftid==0` will be taken. Iteration over all tasks on the processing element will be conducted with the iteration variable `i`, and each task that has issued a `sact[REA]` signal will be checked for unresolved dependencies (the function `dep(t)` checks whether task `t` has unresolved dependencies). If no unresolved dependencies are present for the task at hand, it is deemed ready and entered in the array `tasksReady`. If the task currently has unresolved dependencies, it is placed in the array `wtdep`.

A sact[FIN] signal

If a task has issued a `sact[FIN]` signal, the variable `ftid` will be set to the task's global identification number (which will be different from 0), and the transition guarded by `ftid!=0` is taken. The task is removed from the `tasksReady` array and any tasks depending on it have their dependencies removed with the function `opdDep(t)`, which resolves any current dependencies on task `t`. It then iterates through all tasks to check whether the call to `opdDep(t)` has resolved any dependencies. If any tasks had unresolved dependencies before this call (i.e. were located in `wtdep`), and these dependencies have now been resolved (i.e. the call to `dep(t)` returns true), these tasks are now removed from `wtdep` and added to `tasksReady`. Also, the flag `depCh` is set to make sure a reschedule (`resch`) is broadcasted to all other processing elements. If one or more dependencies have been resolved, the `resch` is issued, but only after making sure that no other tasks are currently finished and waiting to issue their `sact[FIN]` signal; this is ensured using the locking mechanism `h_fin`.

3.4.3 Scheduler

The scheduler makes the decision of which task should be granted processing time on the processing element. This decision is made in observance of a given scheduling principle (e.g. rate monotonic, earliest deadline first, etc.). No matter whether the scheduling principle is statically or dynamically maintained, a scheduling decision is made by checking which task currently has the "highest priority". Since the synchronizer has already handled issues regarding dependencies, the scheduling decision becomes an easy task. In Figure 3.9 the timed automaton for a scheduler is given.

The instantiation of a scheduler specifies which scheduling principle is used.

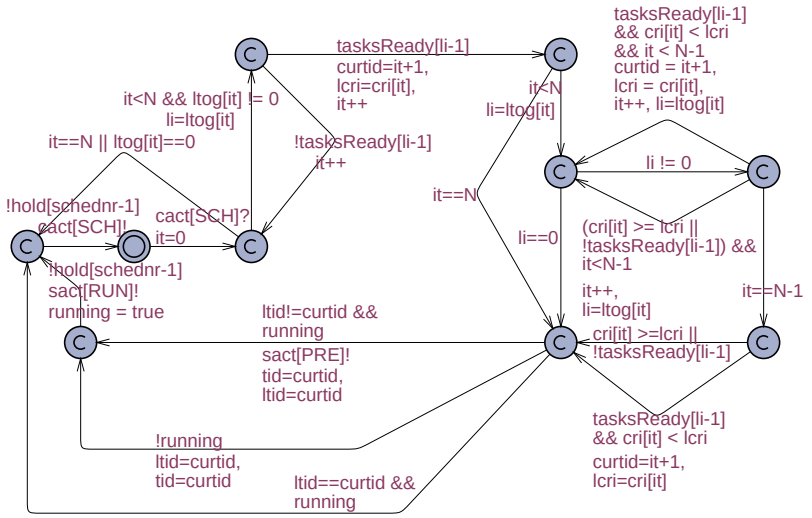


Figure 3.9: Semantics for the Scheduler

The scheduler makes scheduling decisions based on the contents of an array; if this array contains the periods of the tasks in question, the principle is rate monotonic; if the array contains deadline information on the tasks, the principle is earliest deadline first, and so on.

When the scheduler is started, it first checks to see whether any tasks are ready. This is done by checking whether the synchronizer has set any tasks in the `tasksReady` array to true. If a ready task is found, the rest of the `tasksReady` array is inspected and any other ready tasks' criteria are compared to the one currently with the highest priority. Once the full array has been inspected, the ready task with the highest priority has been found. Now scheduling is done by issuing `sact[PRE]` and `sact[RUN]` as needed.

3.5 Discussion

With this semantics in hand, many of the implicit notions and issues from ARTS have been made explicit. In section 5.3.6 an example of a MPSoC is specified and the full timed-automata model for this example can be found in appendix E.

A MPSoC modelled with this semantics contains formal explanations of communication and timing issues. Also, three suggestions to the structure of ARTS have come up:

1. Addition of a controller to govern the communication between the platform and the application. This means that this communication can be formally specified and also that the synchronizer and scheduler can deal with issues specific to these areas only. See section 3.2.2 for more details.

2. Removal of the preemption state. This comes from the realization that a preempted task in reality is just a ready task that has already had some execution time on a processor but is still not finished. This becomes even more evident when it is realized that the semantics for *Preempted* is identical to *Ready*. See section 3.3.6 for further details.

3. Inclusion of an extra transition from *Running* to *Idle* in the ARTS task model for handling the special case where a task finishes in the last time unit of its period and should issue both a `finish` and a `ready` signal. See section 3.3.1 for more details.

Chapter 4

UPPAAL Model of MPSoC Framework

Representing the semantics in the UPPAAL model of timed automata provides some very useful tools and features.

Formal explanation The model of a MPSoC given by the semantics in itself provides a thorough and formal explanation of the system that ARTS itself only describes informally or at implementation level (in SystemC code).

Simulation As ARTS itself is a simulation framework, it is not necessarily a great feature to have the simulation feature of UPPAAL available. However, simulation based on a formal model provides even more insight into the simulation as the simulation is done on a formal basis.

Verification Verification through model checking is the feature that is most revolutionary in providing a semantics through the UPPAAL model. Issues of state explosion and complexity, which make verification of larger systems a timely task, are problematic. But having a model is a first step; then further development in making the model more suitable for verification can be explored.

4.1 Simulation of Formal Model

The simulation engine of UPPAAL is very much self-explanatory. In any state, a choice between the enabled transitions of the timed automata can be taken, and the resulting state is given. UPPAAL also provides a randomized simulation. This means that in every state, one of the enabled transitions is randomly selected. A mixture of manual and random simulation can also be conducted.

Figure 4.1 provides the left-hand side of a screenshot - including transitions, trace and values of variables - of the initial state of an example of an UPPAAL model of a simple system. Figure 4.2 contains the right-hand side of this screenshot with the timed automata and their current states. This figure has been given for the reader to get an understanding of the structure of the simulation. For details on each automaton, please refer to section 3 and appendix E.

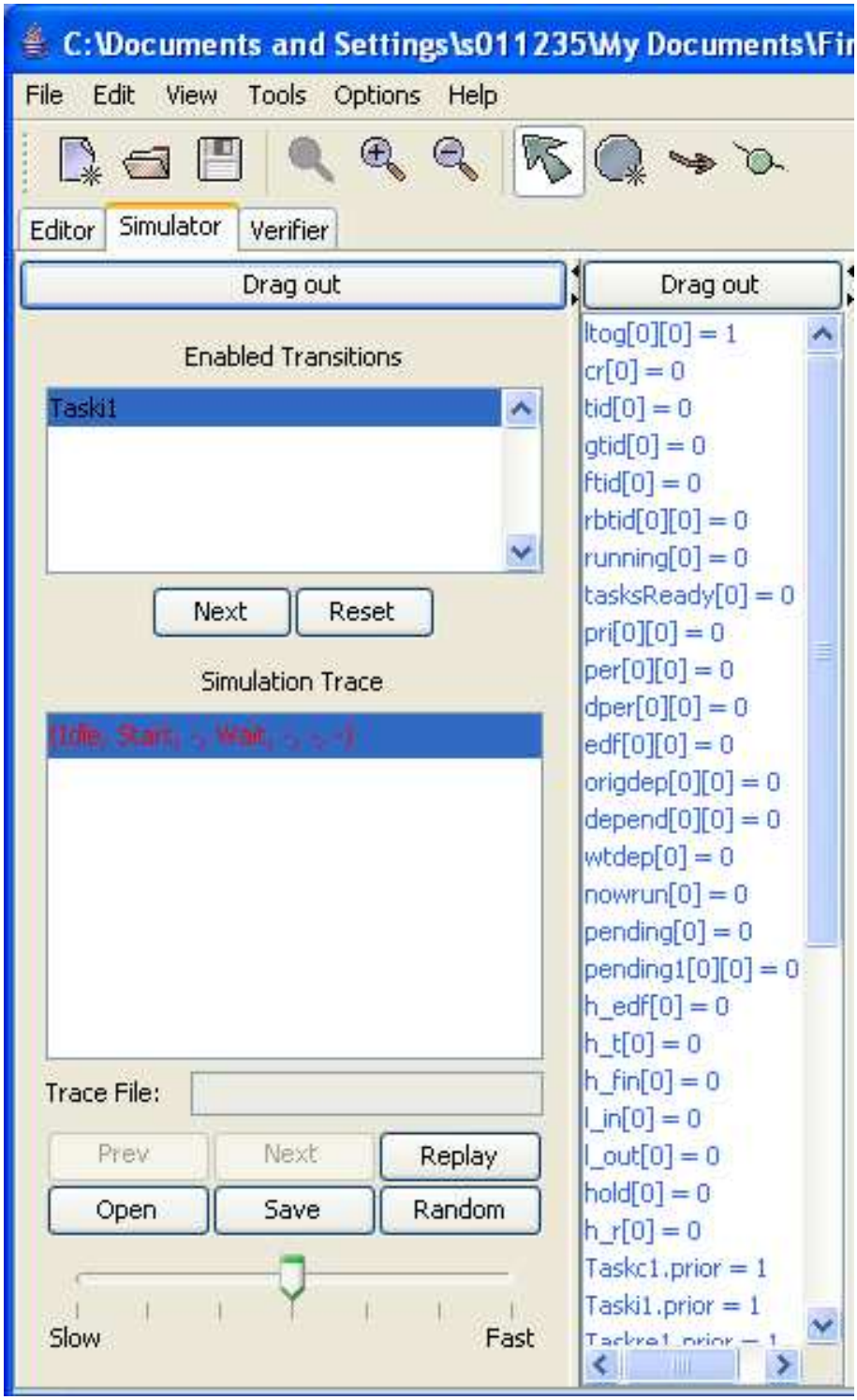


Figure 4.1: Left-hand side of screenshot of initial state

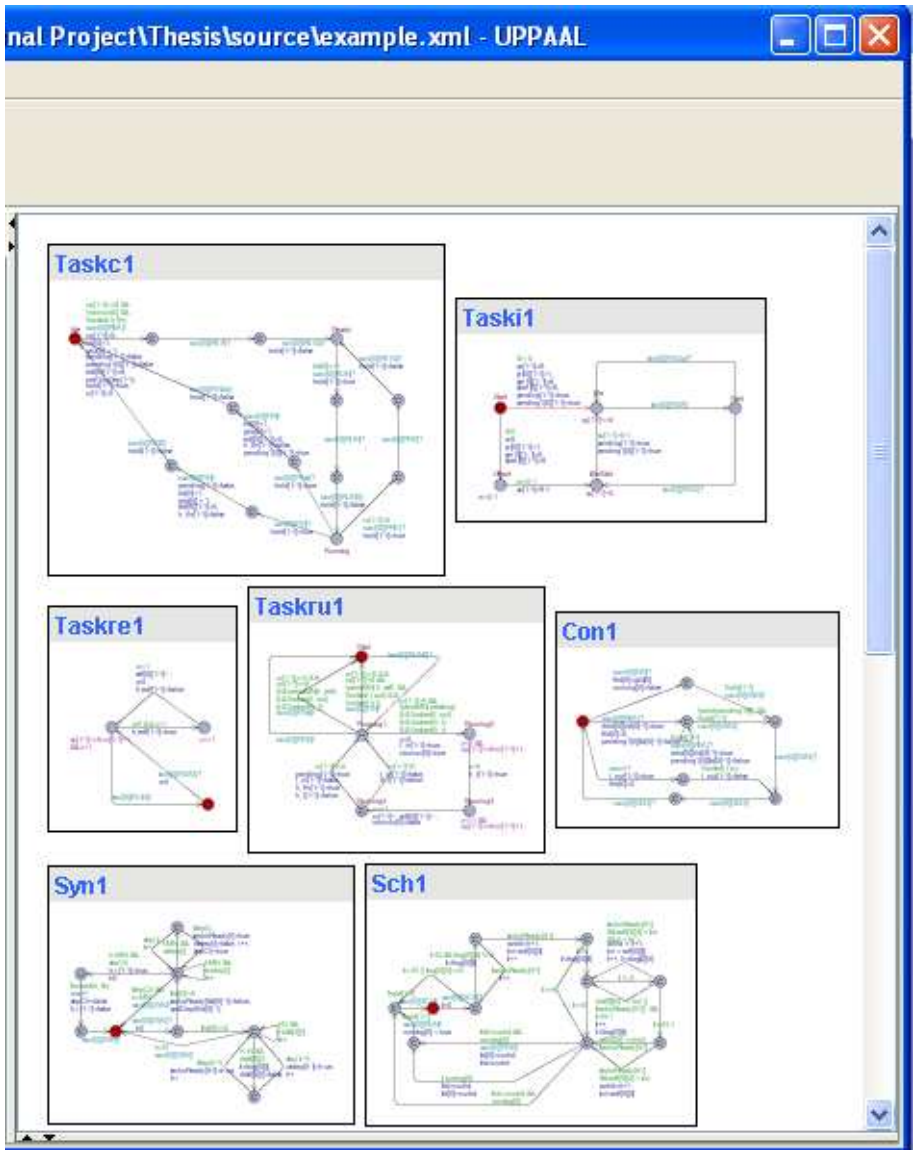


Figure 4.2: Right-hand side of screenshot of initial state

Figure 4.3 contains the left-hand side of a screenshot of the same example as before; however, now some manual steps of the simulation have been conducted. Figure 4.4 is the right-hand side of the screenshot.

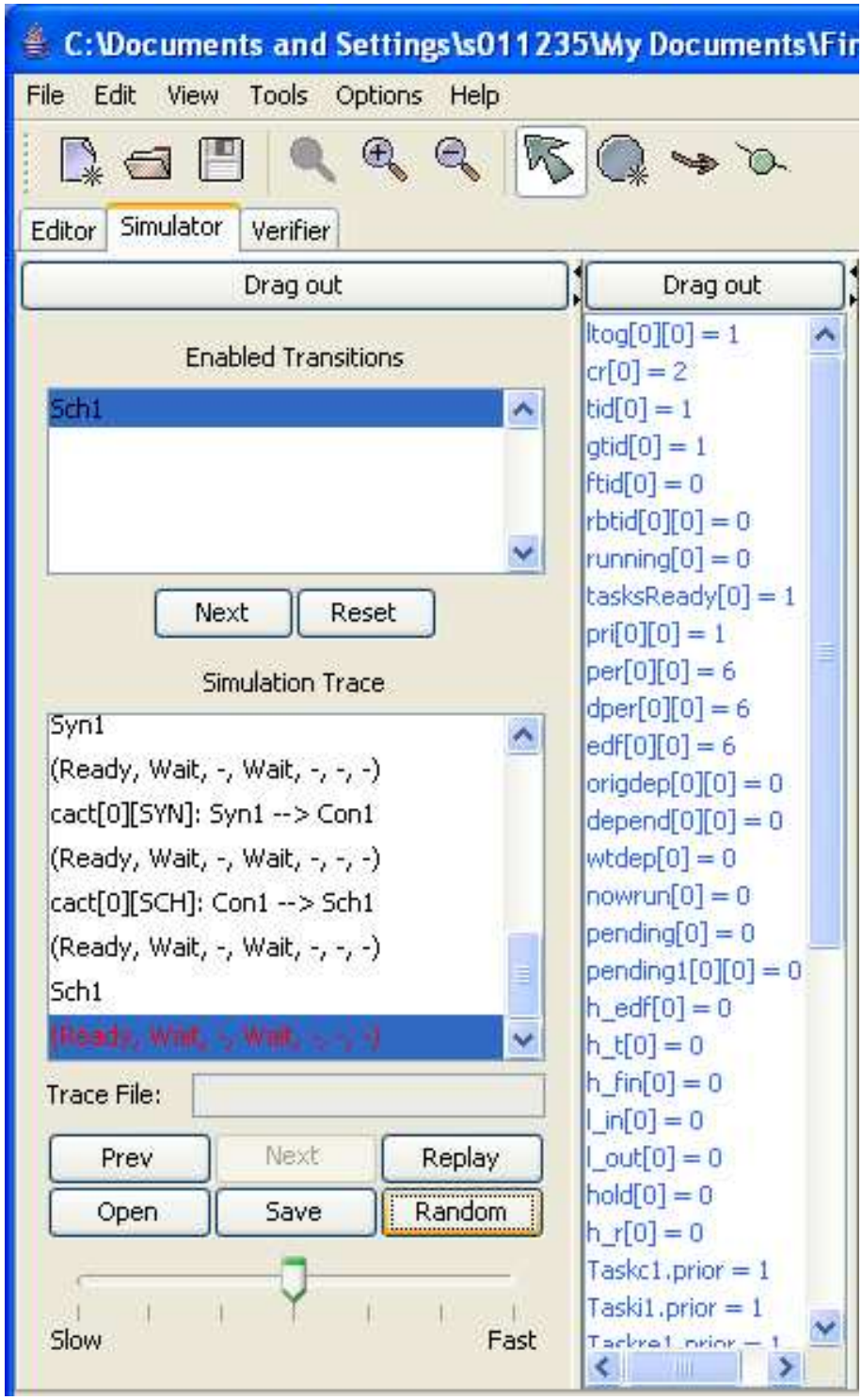


Figure 4.3: Left-hand side of screenshot after some progress

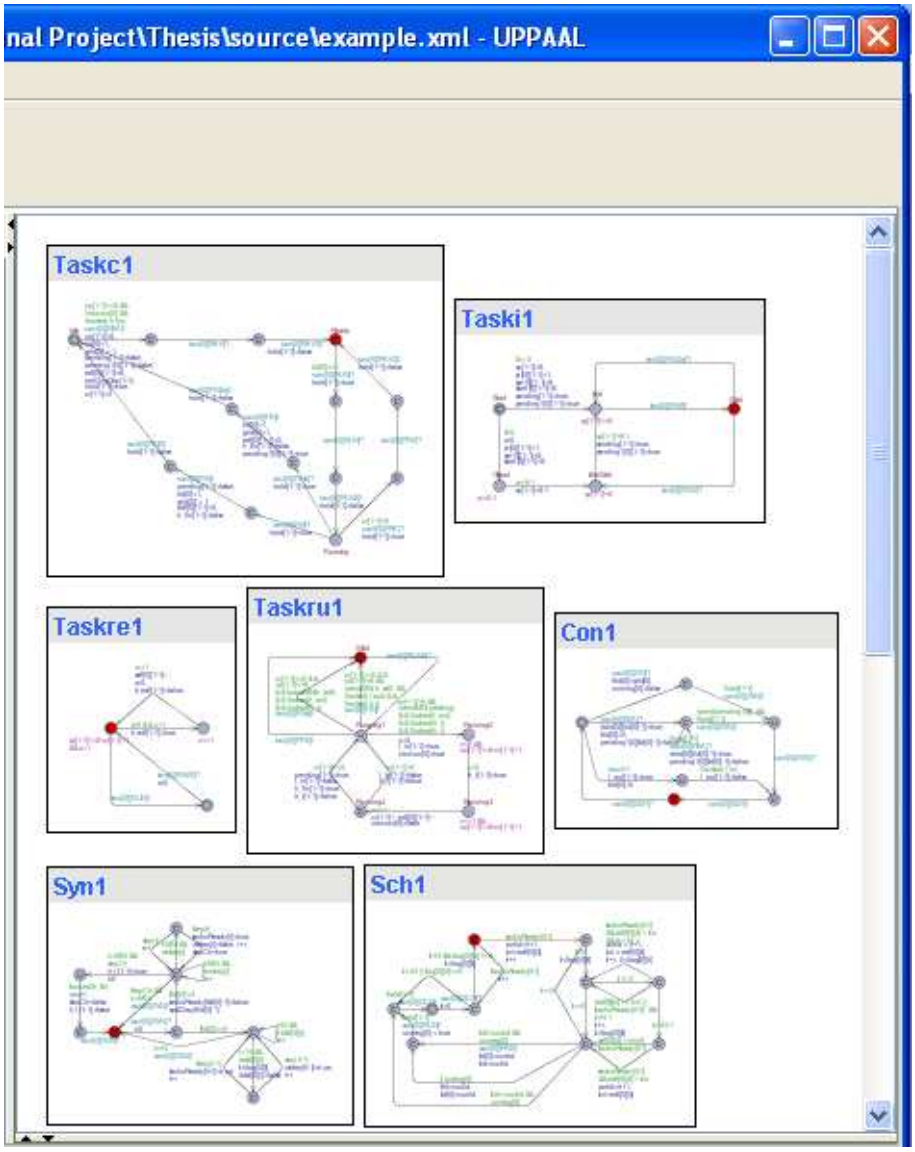


Figure 4.4: Right-hand side of screenshot after some progress

4.2 Verification/Model Checking

As mentioned, verification is the goal for providing this semantics. Guarantees at system level, taking all layers (hardware, middleware and software) into account is a very desirable property. In the semantics provided here, the main focus has been schedulability (i.e. all deadlines are made). Schedulability of a MPSoC modelled in this semantics can be examined by checking for the existence of deadlocks in the system. In UPPAAL this is done with the following query:

$$E <> deadlock$$

If this query returns *Property is satisfied*, a deadlock is detected and the system is not schedulable. If it returns *Property is not satisfied*, no deadlocks are detected and the system is schedulable.

Figure 4.5 provides a screenshot from UPPAAL after a verification has been completed. Note that the response to the verification is *Property is not satisfied*, i.e. the system is schedulable.

In further development the possibility of verification of other properties would be desirable.

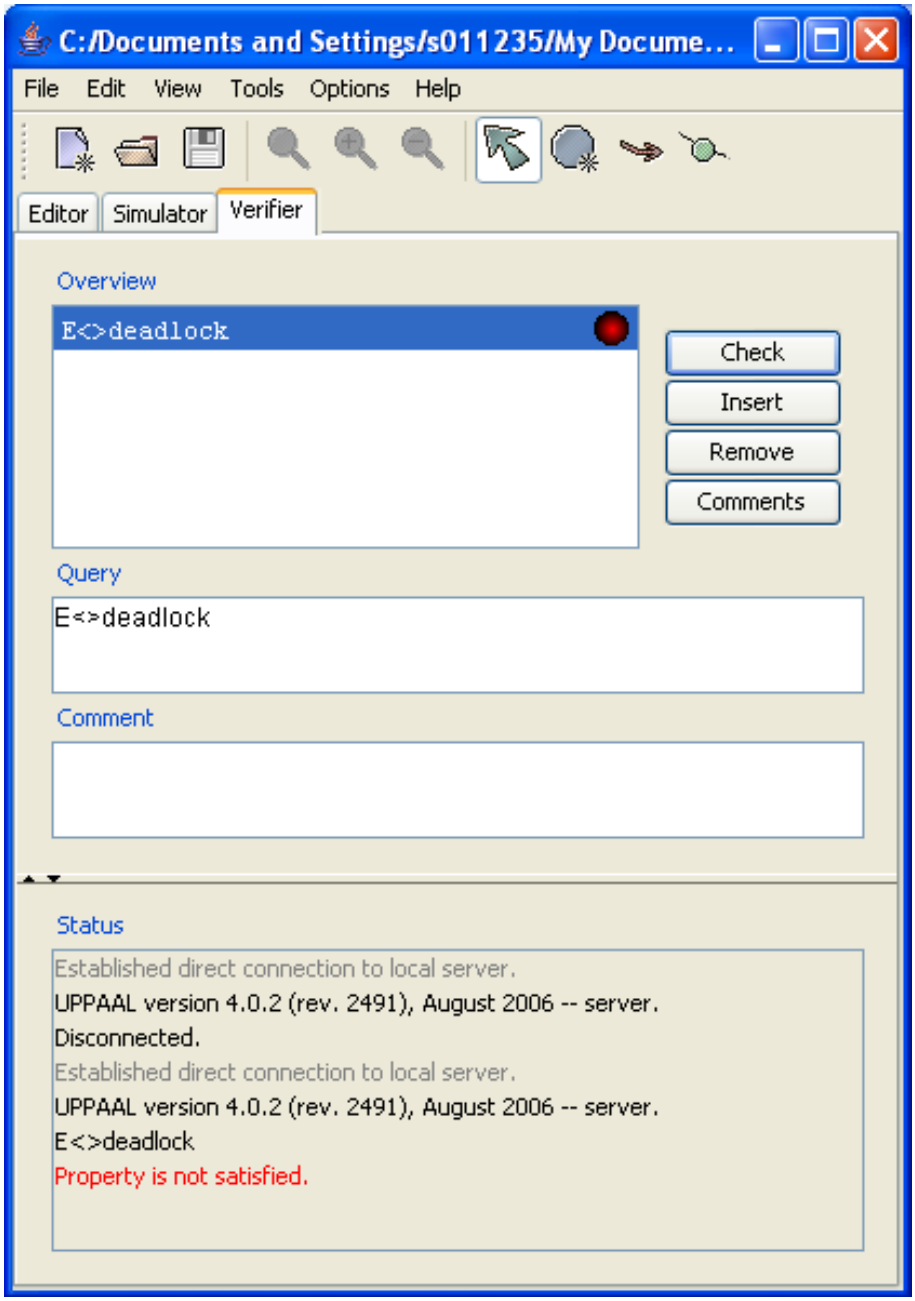


Figure 4.5: Screenshot after verification

Chapter 5

Java Frontend

The input to the UPPAAL system is an XML file, or a model can be manually defined in the graphical user interface. The XML file, as well as the actual representation in UPPAAL with templates, global and local declarations, clocks, variables and system declaration, can seem a bit chaotic. In order to gain a clear picture of what is actually going on in the model, a small client in Java was developed. This client will, based on objects representing the application and the platform as well as dependencies among the different tasks, create the actual XML file, which can then be opened directly in UPPAAL. In the following, a description of the Java frontend will be given, its requirements and how it was implemented will be explained, and small examples of how to use this tool will be provided.

5.1 Requirements

The overall purpose of this client is to generate the XML file, which can be used as input for the UPPAAL system. It is, however, desired that the structure of the objects that make up the client depicts the structure described in ARTS and in this thesis. Therefore, objects representing tasks and processing elements are the base objects, and these can then be collected in application and platform

objects - the application object also contains dependency information. Finally, a MPSoC object is described by the application and the platform objects.

5.2 Implementation

In Figure 5.1 a class diagram showing the structure of the Java client is given.

The classes *Task*, *Processor*, *Platform* and *Application* make up the building blocks for the system. They are gathered in the *MPSoC* class that provides the method `mkXML()`, which produces a `String` containing the input to UPPAAL.

In appendix B the full source code in Java can be found.

5.3 How to use

Using the Java client is very simple. First the tasks and processors are defined. Then the platform is defined by the processors and the application by the tasks. Then possible dependencies among the tasks are expressed in the application. Finally, the MPSoC is defined by the application and the platform and the method `mkXML()` can be invoked, resulting in a `String` representation for the input to UPPAAL.

5.3.1 Task

A task is instantiated with values for *execution time*, *deadline*, *offset*, *period*, *user defined priority* and *processor identification*. Examples of three tasks are given here. The task t_1 has execution time 2, deadline and period 6, offset 0, priority 1 for user defined priority scheduling and is mapped to processor 1. The task t_2 has execution time 2, period 10 and deadline 9, offset 2, priority 2 for user defined priority scheduling and is mapped to processor 2. Finally, the task t_3 has execution time 3, deadline and period 15, offset 0, priority 3 for user defined priority scheduling and is mapped to processor 2. The instantiation of these three tasks is performed as follows:

```
Task t1 = new Task(2, 6, 0, 6, 1, 1);
```

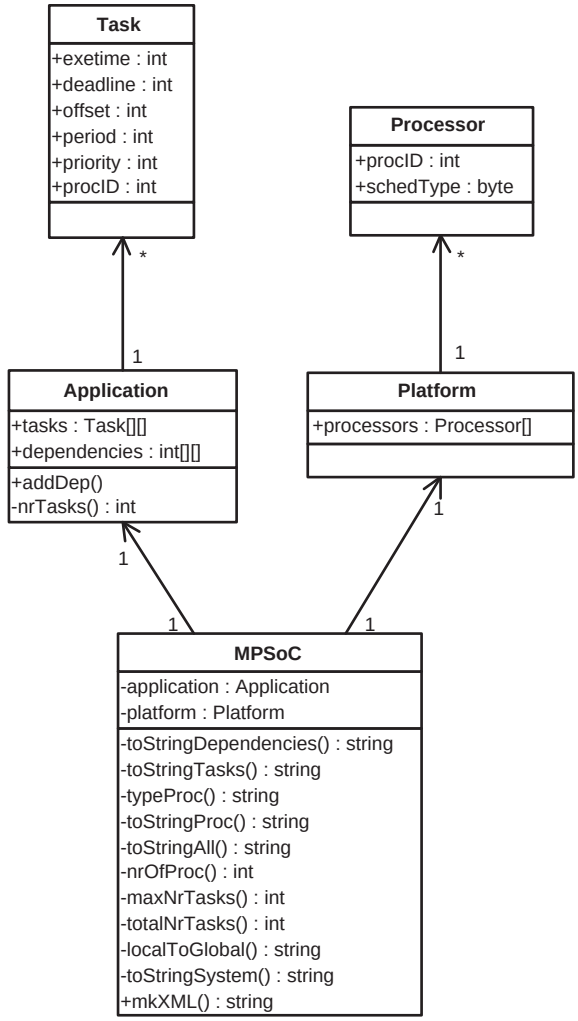


Figure 5.1: Class diagram for Java client

```
Task t2 = new Task(2, 10, 0, 9, 2, 1);

Task t3 = new Task(3, 15, 0, 15, 3, 2);
```

5.3.2 Processor

A processor is instantiated with values for the *processor identification* and *scheduling principle*. The currently implemented principles are:

Scheduling Principle	Java value
Earliest deadline first	EDF
Rate monotonic	RM
Deadline monotonic	DM
User defined fixed priority	FP

A processor (p_1) with identification number 1 and earliest deadline first scheduling as well as a processor (p_2) with identification number 2 and rate monotonic scheduling are specified as follows:

```
Processor p1 = new Processor(1, Processor.EDF);

Processor p2 = new Processor(2, Processor.RM);
```

5.3.3 Application

The application is instantiated by a double array containing the tasks. An application containing tasks t_1 and t_2 mapped to one processor and t_3 on another is specified as follows:

```
Application app = new Application({t1,t2},{t3})
```

Dependencies can now be expressed with the method `addDep(int,int)`, which expresses a dependency from one task to another. For example, if t_3 needs to wait for completion of t_2 (i.e. there is a dependency from t_2 to t_3) this is expressed as follows:

```
app.addDep(2,3);
```

5.3.4 Platform

The platform is instantiated by an array of processors making it up. A platform with the processors p_1 and p_2 is specified as follows:

```
Platform pl = new Platform({p1,p2});
```

5.3.5 MPSoC

Finally, the MPSoC is instantiated by its application and platform. The MPSoC made up by the above-mentioned application (`app`) and platform (`pl`) is specified as follows:

```
MPSoC mps = new MPSoC(app,pl);
```

A string representation of the input to UPPAAL for this MPSoC is generated by the following invocation:

```
mps.mkXML()
```

5.3.6 Defining the full system

The system that is described in pieces in this chapter can be explained by the following table of timing constraints and the task graph in Figure 5.2.

Task #	Execution time	Period	Deadline	Offset	Processor #
1	2	6	6	0	1
2	2	10	9	2	1
3	3	15	15	0	2

The following Java program represents the system and creates the file `example.xml` that serves as input to UPPAAL and models the system. This system can then be simulated and verified in UPPAAL. The Java source code has been commented here; however, in the following code examples, comments have been left out.

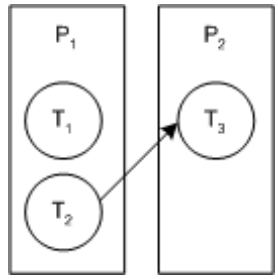


Figure 5.2: Task graph for running example in this chapter

```
import java.io.*;

public class example{
    public static void main(String[] args){
        //file name before the .xml extention
        String fl = "example";

        //Declaration of tasks
        Task t1 = new Task(2, 6, 0, 6, 1, 1);
        Task t2 = new Task(2, 10, 2, 9, 2, 1);
        Task t3 = new Task(3, 15, 0, 15, 3, 2);

        //Declaration of processors
        Processor p1 = new Processor(1, Processor.EDF);
        Processor p2 = new Processor(2, Processor.RM);

        //Gathering the tasks
        Task[][] tasks = { { t1, t2} , { t3} };

        //Gathering the processors
        Processor[] ps = {p1,p2};

        //Declaration of the platform
        Platform platform = new Platform(ps);

        //Declaration of the application
        Application apps = new Application(tasks);

        //Adding dependencies
        apps.addDep(2,3);

        //Declaration of the MPSoC
        MPSoC sys = new MPSoC(apps,platform);

        //Making the XML file
        try{
            PrintStream fout = new PrintStream(new File(fl+".xml"));
            fout.println(sys.mkXML());
        }
        catch(Exception e){e.printStackTrace();}
    }
}
```

The complete model in terms of the timed automata and global and local declarations as well as the system declaration is provided in appendix E.

Chapter 6

Examples

In this chapter a collection of small examples will be given. These will show how important notions from ARTS can be modelled and allow for simulation and verification in UPPAAL. It will be explained how they have been expressed in terms of Java programs, which generate the XML file UPPAAL uses as input.

First, some single processor examples, both with and without dependencies, will be explained and the results from the verification will be given. Where appropriate, these results will be compared to schedulability analyses using TIMES tool [2] on the same examples, to provide some evidence for the correctness of the verification.

Then some multiprocessor systems will be given and the results explained. Some of these are based on examples given in the literature; again, this will provide evidence for the correctness of the verification. The issue of inter-processor dependencies will also be explored.

Many of the examples (both for single processor as well as multiprocessor systems) will examine design issues regarding the operating systems on the processing elements and exemplify how introducing a different scheduling principle can sometimes make scheduling possible where it was not possible using the original scheduling principle.

6.1 Single Processor

In this section a collection of single processor examples will be discussed.

Single processor example 1

This simple example with three tasks on a single processor can - according to schedulability analyses using TIMES tool - be scheduled using either rate monotonic or earliest deadline first.

Task #	Execution time	Period	Deadline	Offset
1	6	10	10	0
2	6	20	20	0
3	2	30	30	0

The following Java program creates the file `single_ex_1_rm.xml`, which is the input to UPPAAL representing the system declared in the table above using rate monotonic scheduling.

```
import java.io.*;

public class single_ex1_rm{
    public static void main(String[] args){
        String fl = "single_ex1_rm";
        Task t1 = new Task(6, 10, 0, 10, 1, 1);
        Task t2 = new Task(6, 20, 0, 20, 2, 1);
        Task t3 = new Task(2, 30, 0, 30, 3, 1);

        Processor p1 = new Processor(1, Processor.RM);

        Task[][] tasks = { { t1,t2,t3} };

        Processor[] ps = {p1};
        Platform platform = new Platform(ps);
        Application apps = new Application(tasks);

        MPSoC sys = new MPSoC(apps,platform);
        try{
            PrintStream fout = new PrintStream(new File(fl+".xml"));
            fout.println(sys.mkXML());
        }
        catch(Exception e){e.printStackTrace();}
    }
}
```

Verification of the UPPAAL model returns *Property is not satisfied* - i.e. no deadlocks in the system. This means that the system is schedulable.

The system is now modelled using earliest deadline first scheduling; the corresponding Java program can be seen below.

```
import java.io.*;

public class single_ex1_edf{
    public static void main(String[] args){
        String fl = "single_ex1_edf";
        Task t1 = new Task(6, 10, 0, 10, 1, 1);
        Task t2 = new Task(6, 20, 0, 20, 2, 1);
        Task t3 = new Task(2, 30, 0, 30, 3, 1);

        Processor p1 = new Processor(1, Processor.EDF);

        Task[][] tasks = { { t1,t2,t3} };

        Processor[] ps = {p1};
        Platform platform = new Platform(ps);
        Application apps = new Application(tasks);

        MPSoC sys = new MPSoC(apps,platform);
        try{
            PrintStream fout = new PrintStream(new File(fl+".xml"));
            fout.println(sys.mkXML());
        }
        catch(Exception e){e.printStackTrace();}
    }
}
```

The execution of the program produces the file `single_ex_1_edf.xml`, which is the input to UPPAAL representing the system using earliest deadline first scheduling.

Verification of this UPPAAL model also returns *Property is not satisfied*.

In the following, only the tables showing the timing constraints for the systems will be given. The Java programs representing these single processor systems can be found in appendix C.

Single processor example 2

This example closely resembles *Single processor example 1* with the small difference that task 3 has one more time unit of execution. This means that the system is now not schedulable using rate monotonic scheduling but is schedulable using earliest deadline first according to schedulability analyses using TIMES tool.

Task #	Execution time	Period	Deadline	Offset
1	6	10	10	0
2	6	20	20	0
3	3	30	30	0

Verification of the UPPAAL model of this system using rate monotonic scheduling returns *Property is satisfied*. This means that there is one or more deadlocks in the system.

Using earliest deadline first scheduling, verification of the UPPAAL model returns *Property is not satisfied*. In other words, the system is schedulable using earliest deadline scheduling but not using rate monotonic.

Single processor example 3

This example resembles both of the previous examples. But this time the execution time of task 3 has been extended by one time unit so that scheduling - according to TIMES tool - cannot be done using either rate monotonic or earliest deadline first scheduling.

Task #	Execution time	Period	Deadline	Offset
1	6	10	10	0
2	6	20	20	0
3	4	30	30	0

Verification of the UPPAAL model of this system using first rate monotonic and then earliest deadline first scheduling both return *Property is satisfied*.

Single processor example 4

This example is given to show how deadline monotonic scheduling sometimes can be used instead of rate monotonic in order to make a system schedulable. Running schedulability analyses in TIMES tool provides this result.

Task #	Execution time	Period	Deadline	Offset
1	3	5	5	0
2	2	4	6	0

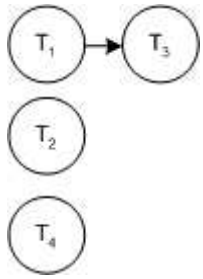


Figure 6.1: Task graph for single processor example 5

Verification of the UPPAAL model of the system using rate monotonic scheduling returns *Property is satisfied*, and using deadline monotonic scheduling, *Property is not satisfied* is returned.

Single processor example 5

In the following example a system with intra-processor dependencies are examined. It should be noted that resolving dependencies in TIMES is done differently than in ARTS. Therefore, results of schedulability analyses of systems having dependencies in TIMES cannot be directly compared to verification of the UPPAAL model of these systems. In this example, scheduling is possible using earliest deadline first scheduling but not rate monotonic (TIMES tool says scheduling is possible also with rate monotonic scheduling due to a different interpretation on how to resolve dependencies). In the system there is a dependency from task 1 to task 3; in other words, task 1 must finish execution before task 3 can start execution. This is depicted by the task graph in Figure 6.1

Task #	Execution time	Period	Deadline	Offset
1	3	15	15	0
2	2	10	10	0
3	4	35	35	0
4	5	13	13	0

Verification of the UPPAAL model of this system using rate monotonic scheduling returns *Property is satisfied*. With earliest deadline first, verification returns *Property is not satisfied*.

6.2 Multiprocessor

A collection of multiprocessor examples will be shown and the results explained and analyzed. All of the examples are systems with dependencies, as systems without dependencies basically can be seen as individual single processor systems. Figures depicting the task graphs for the systems will also be given.

Multiprocessor example 1

This system is based on an example from Sun and Liu [16]. The system cannot be scheduled using rate monotonic scheduling but is schedulable using earliest deadline first. Figure 6.2 shows how task 3 is dependant on task 2.

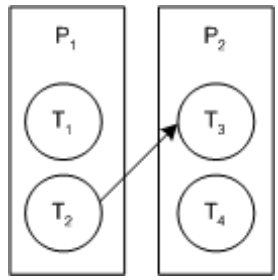


Figure 6.2: Task graph for multiprocessor example 1

Task #	Execution time	Period	Deadline	Offset	Processor #
1	2	4	4	0	1
2	2	6	6	0	1
3	2	6	6	0	2
4	3	6	6	4	2

The following Java program represents the system using rate monotonic scheduling and produces the file `multi_ex1_rm`, which acts as input to UPPAAL.

```
import java.io.*;

public class multi_ex1_rm{
    public static void main(String[] args){
```

```
        String fl = "multi_ex1_rm";
        Task t1 = new Task(2, 4, 0, 4, 1, 1);
        Task t2 = new Task(2, 6, 0, 6, 2, 1);
        Task t3 = new Task(2, 6, 0, 6, 3, 2);
        Task t4 = new Task(3, 6, 4, 6, 4, 2);

        Processor p1 = new Processor(1, Processor.RM);
        Processor p2 = new Processor(2, Processor.RM);

        Task[][] tasks = { { t1, t2} , { t3, t4 } };

        Processor[] ps = {p1,p2};
        Platform platform = new Platform(ps);
        Application apps = new Application(tasks);

        apps.addDep(2,3);

        MPSoC sys = new MPSoC(apps,platform);
        try{
            PrintStream fout = new PrintStream(new File(fl+".xml"));
            fout.println(sys.mkXML());
        }
        catch(Exception e){e.printStackTrace();}
    }
}
```

Verification of the UPPAAL model of this system returns *Property is satisfied*.

The following Java program represents the system using earliest deadline first scheduling and produces the file `multi_ex_edf`.

```
import java.io.*;

public class multi_ex1_edf{
    public static void main(String[] args){
        String fl = "multi_ex1_edf";
        Task t1 = new Task(2, 4, 0, 4, 1, 1);
        Task t2 = new Task(2, 6, 0, 6, 2, 1);
        Task t3 = new Task(2, 6, 0, 6, 3, 2);
        Task t4 = new Task(3, 6, 4, 6, 4, 2);

        Processor p1 = new Processor(1, Processor.EDF);
        Processor p2 = new Processor(2, Processor.EDF);

        Task[][] tasks = { { t1, t2} , { t3, t4 } };

        Processor[] ps = {p1,p2};
        Platform platform = new Platform(ps);
        Application apps = new Application(tasks);

        apps.addDep(2,3);

        MPSoC sys = new MPSoC(apps,platform);
        try{
            PrintStream fout = new PrintStream(new File(fl+".xml"));
            fout.println(sys.mkXML());
        }
        catch(Exception e){e.printStackTrace();}
    }
}
```

Verification of this UPPAAL model returns *Property is not satisfied.*

In the following multiprocessor examples, tables giving the timing constraints and task graphs for the systems will be shown. Java programs representing the systems can be found in appendix D.

Multiprocessor example 2

This system is, just like multiprocessor example 1, based on the example from Sun and Liu [16]. Task 3 in this example basically represents the communication between the two processors; in other words, the processor is more like a "virtual" processor representing the shared medium over which the two "real" processors communicate, and task 3 is a "virtual" task representing the actual communication done on the shared medium. The system's dependencies are depicted on the task graph in Figure 6.3 and can, just like multiprocessor example 1, be scheduled using earliest deadline first scheduling but not using rate monotonic.

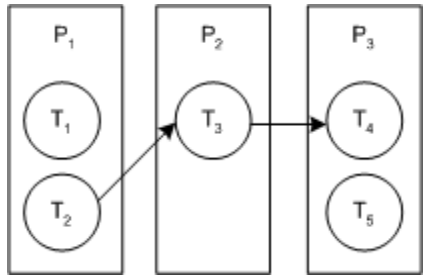


Figure 6.3: Task graph for multiprocessor example 2

Task #	Execution time	Period	Deadline	Offset	Processor #
1	2	4	4	0	1
2	1	6	6	0	1
3	1	6	6	0	2
4	2	6	6	0	3
5	3	6	6	4	3

Verification of the UPPAAL model of the system has the same results as with multiprocessor example 1.

Multiprocessor example 3

This example was included to show what size systems the model is able to verify. The following example was verified in less than 2 minutes on a 3.4GHz PC with 1GB of Ram running UPPAAL version 4.0.2. The task graph for the system can be seen in Figure 6.4.

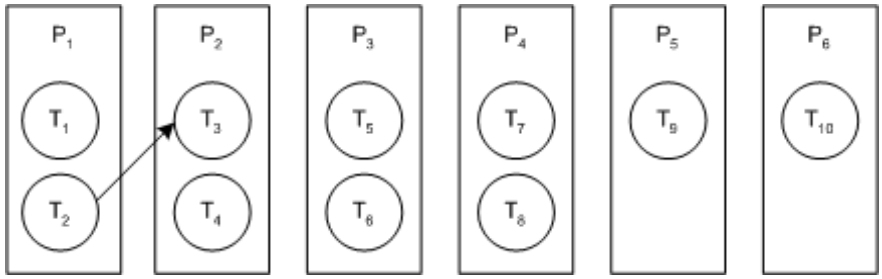


Figure 6.4: Task graph for multiprocessor example 3

Task #	Execution time	Period	Deadline	Offset	Processor #
1	2	40	40	0	1
2	2	40	40	0	1
3	2	40	40	0	2
4	2	40	40	0	2
5	2	40	40	0	3
6	2	40	40	0	3
7	2	40	40	0	4
8	2	40	40	0	4
9	2	40	40	0	5
10	2	40	40	0	6

Verification of the UPPAAL model of the system returns *Property is not satisfied* when using earliest deadline first, rate monotonic or deadline monotonic scheduling.

Chapter 7

Future Development

With the model provided in this thesis as a basis, this chapter will give indications as to which directions further work within formal modelling of MPSoC can take. There will be suggestions of inclusions of a larger subset of the ARTS model and how these can be started, as well as suggestions of how the model can be adapted to verify and optimize on other costs besides the timing described in the current model. Also, other areas where this type of model could be useful will be noted.

7.1 Expanding areas of ARTS included

In this thesis a model for MPSoC has been given. This model takes the ARTS system as a basis and formalizes the informal notions present in the ARTS model. The current model can represent a significant subset of systems that can be modelled in ARTS. The following are some of the properties of systems that are included in the model:

- Periodic tasks
- Processing elements consisting of synchronizer and scheduler elements

- Verification of scheduling in regard to timing constraints

In ARTS more advanced features for MPSoC can be modelled and simulated; inclusion of some of these would be an appropriate enhancement of the model. These features include the following:

- Non-periodic tasks
- Inclusion of resource allocation
- Modelling of communication between different processing elements (i.e. different communication media - busses, networks, etc.)
- Modelling of memory and power usage

7.2 Introducing cost functions in priced timed automata

Modern design of MPSoC is not only reliant on timing constraints. It would be appropriate to include properties additional to timing constraints of a MPSoC for verification. Using models for priced timed automata, it would be possible to add other costs besides timing. These could be costs such as memory and energy usage. A model including such criteria may enable verification based on factors other than time, and with priced timed automata, optimization with regard to these criteria could be introduced.

7.3 Other areas where the model can be useful

From an overall perspective, the current model basically deals with a complex system made up of different distributed parts or subsystems. Verification of these mean checking for deadlines of the system that are missed. A model like this could be used in many cases that do not directly relate to the domain described there (MPSoC). Introducing cost (e.g. in terms of power usage) with priced timed automata would enable checking for optimization with regard to this cost. Another area where this approach could prove useful would be in optimizing distributed systems with regard to delay. In short, the model at hand is used in a specific domain; however, introducing this type of model in other domains could provide the same type of verification possibilities as described for MPSoC in this thesis.

Chapter 8

Conclusion

A formal model for MPSoC using ARTS as a basis has been developed using timed automata. The model supports simulation and verification using UPPAAL models of timed automata.

Initially it has been specified how ARTS, with its modular structure, provides a useful basis for a formal model that describes MPSoC in subcomponents which, combined in the same modular structure, formalize the overall system in full.

The full formal model provides mathematical meaning to the MPSoC. This model provides a greater understanding of each MPSoC modelled and opens up for simulation and verification on a formal basis. Modelling, simulation and verification of timed automata are easily performed in UPPAAL through its graphical user interface.

Using this formal model, examples verified in other places (i.e. using TIMES tool) and additional examples from the literature (i.e. [16]) that have known performance properties can be captured.

To be able to specify systems in a simple way without direct adaptation of the formal model, a Java frontend has been developed. With this, systems can be easily modelled in terms of UPPAAL models of timed automata.

Verification in UPPAAL provides results corresponding to those from the literature and analyses using other tools. This provides evidence as to the correctness of the models.

The formal model adds an extra level of abstraction to MPSoC and verification can be used to assist in analysis of correctness and performance parameters of such systems. This analysis can be of great use in different development stages of the design phase of MPSoC.

Beyond showing how a formal model can be developed and thereby answering this thesis' problem formulation, the work completed in the project leading up to this thesis has shown that the formalization of the model can produce feedback for the original model in areas where the original model is inaccurate or does not capture certain special cases (e.g. the case when a task finishes in the same time unit as its next period starts). The original model did not capture these; however, the formal model does. Some suggestions to the overall structure of the model have also been given as a result of the clarity the formal model provided, e.g. the introduction of a controller component.

The project has been successful in providing a model that not only provides results to the domain but can be used as a basis in further development where additional functionalities and properties of systems can be included, such as memory and power usage and communication principles.

Appendix A

Introduction to timed automata

This introduction to timed automata is inspired by lectures and lecture notes on Process and Data Modelling by Jens Christian Godskesen, IT University of Copenhagen and on Real-Time Systems by Jesper Blak Møller, Technical University of Denmark.

In the following, it is assumed that the reader has general knowledge of *finite automata*.

A.1 Finite automata

An example of an electric door with sensors will be used to exemplify the extensions to finite automata.

We define a *finite automaton* M as a five tuple:

$$M = (Q, \Sigma, \rightarrow, q_0, F)$$

where

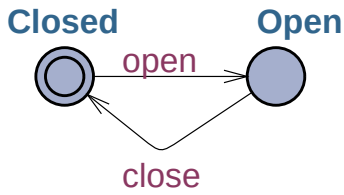


Figure A.1: A finite automaton for the electric door

- Q is a set of *states*
- Σ is a finite set of *input symbols*
- $\rightarrow \subseteq Q \times \sigma \times Q$ is the *transition relation*
- q_0 is the *initial state*
- $F \subseteq Q$ is the set of accepting states

If $(q, a, q') \in \rightarrow$ we say that there is a transition from q to q' written as $q \xrightarrow{a} q'$

Door example - finite automata

Consider an electric door. When in the *Closed* state and given an `open` symbol, it moves to the *Open* state.When in the *Closed* state and given a `close` symbol, it moves to the *Closed* state. A model for the door as a finite automata can be seen in Figure A.1:

A.2 Communicating finite automata

A communicating finite automaton is a finite automaton where each transition is either an output, input or internal transition.

We write:

- $q \xrightarrow{a!} q'$ for an output transition,
- $q \xrightarrow{a?} q'$ for an input transition and

- $q \longrightarrow q'$ for an internal transition.

A communicating finite automaton is defined by (the set of accepting states is omitted from now on as we focus on communication/synchronization only):

$$M = (Q, \Sigma, \rightarrow, q_0)$$

where Q, Σ and q_0 are as before and:

$$\rightarrow \subseteq (Q \times \Sigma \times \{!, ?\} \times Q) \cup (Q \times Q)$$

Communicating finite automata run in parallel and synchronize on input/output symbols.

Let $M_1 = (Q_1, \Sigma, \rightarrow_1, q_1)$ and $M_2 = (Q_2, \Sigma, \rightarrow_2, q_2)$ be communicating finite automata. The composition of M_1 and M_2 is the finite automaton:

$$M = (Q_1 \times Q_2, \Sigma, \rightarrow, (q_1, q_2))$$

where $\rightarrow$ is defined by:

$$\frac{p \xrightarrow{a!}_1 p' \; q \xrightarrow{a?}_2 q'}{(p,q) \xrightarrow{a} (p',q')} \qquad \frac{q \xrightarrow{a!}_2 q' p \xrightarrow{a?}_1 p'}{(p,q) \xrightarrow{a} (p',q')}$$

$$\frac{p \longrightarrow_1 p'}{(p,q) \longrightarrow (p',q)} \qquad \frac{q \longrightarrow_2 q'}{(p,q) \longrightarrow (p,q')}$$

Door example - communicating finite automata

In the electric door example, it is wished to model a sensor as well as the door. The sensor can either have light or no light, modelled by the states *Light* and *NoLight*. When moving from *Light* to *NoLight* an `Obstruct` signal is issued, and when moving from *NoLight* to *Light* an `UnObstruct` signal is issued. The sensor is modelled by the communicating finite automata in Figure A.2. The door is modelled in the communicating finite automaton in Figure A.3. The channels `Obstruct` and `UnObstruct` model whether or not the sensor is obstructed.

A.3 Extended finite automata

A communicating finite automaton may be extended with variables. Execution of a transition may depend on the value of the variables, and values of variables

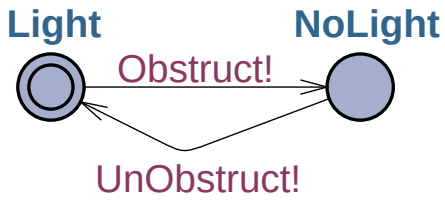


Figure A.2: A communicating finite automaton for the sensor

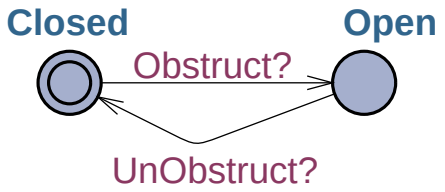


Figure A.3: A communicating finite automaton for the door

may be assigned when performing a transition.

$$p \xrightarrow{a!,x>10,y:=5} p'$$

means that the transition from p to p' can only be performed if the value of x is greater than 10 (the boolean expression $x > 10$ is called a guard), and when performing the transition the value of y is set to 5.

Door example - extended finite automata

We wish to model that the door cannot handle more than 100 openings and closings per day; this is done by extending the model for the door with the variable `count`. It is increased every time the door opens, and the door cannot open unless `count` is less than 100. It is thought to be reset every day. The extended finite automata for the door can be seen in Figure A.4

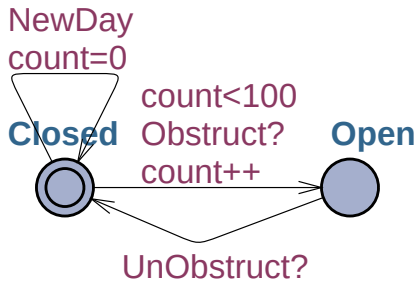


Figure A.4: An extended finite automaton for the door

A.4 Timed finite automata

A timed communicating finite automaton is an extended finite automaton containing real valued clocks. Clocks measure the progress of time and may be part of a guard. The transition:

$$p \xrightarrow{a!,c>10,c:=0} p'$$

can only be performed when the time of clock c is greater than 10, and when the transition is performed the clock is reset. Furthermore, clocks allow for timed invariants on states. A timed invariant specifies when it is possible to be in the state in question. For example, the invariant: $c \leq 10$ specifies that the state cannot be entered unless the clock c is less than or equal to 10. Furthermore, when in the state in question, it must leave before the c exceeds 10.

Door example - timed finite automata

We wish to model that the door is kept open at least 5 seconds after every opening. If the sensor gets obstructed within those 5 seconds, the door must be kept open for another 5 seconds. Therefore, the local clock `x` is introduced in the model. Resetting the variable `count` each day is done by introducing a clock `day`, which, every 86400 seconds, resets `count` and itself in any state of the model. The model for the door as a timed automaton can be seen in Figure A.5.

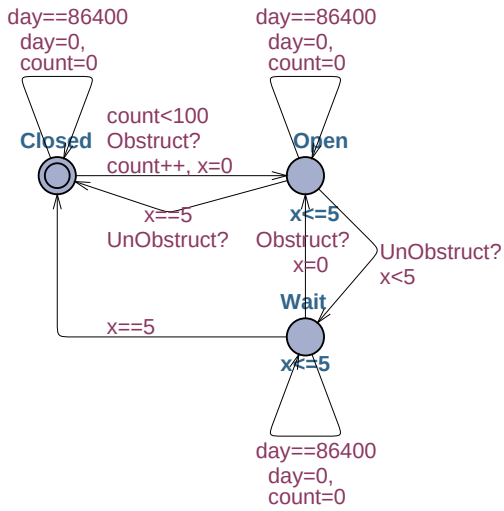


Figure A.5: A timed automaton for the door

A.5 UPPAAL

UPPAAL is a tool with which timed communicating finite automata can be modelled using a graphical editor. Essentially, an UPPAAL model contains:

- global declarations of channels, variables and clocks
- a parameterized template for each type of automaton including local declarations of variables and clocks
- a definition of each automaton based on its template
- a system definition consisting of the automata making up the system

With a system declaration, UPPAAL provides a tool for simulating the system and a tool for verification of properties of the system. The simulation can be conducted either by choosing a random simulation or manually selecting each transition or a combination of both. In a random simulation, UPPAAL randomly chooses the next transition from the different enabled transitions. In a manual simulation, the user can choose each transition from a list of enabled transitions. If we would like to validate whether a certain property is guaranteed by the system, we must (more or less systematically) try to select all possible

paths through the simulation. In real life models it may be an almost impossible task through simulation to make sure that a certain property of a composed system is satisfied. With the help of the verification tool in UPPAAL, we can have the states of a model explored automatically.

Urgent and committed states

UPPAAL provides convenient notions of urgent and committed states. These are explained here:

Urgent An urgent state is a state where time cannot pass. This could also be modelled as follows: a local clock x is reset to 0 on all incoming transitions to the state, and an invariant $x \leq 0$ forces exit of the state before time passes.

Committed Committed state are - like urgent - states where time cannot pass. Furthermore, when at least one timed automaton of a system is in a committed state, only transitions leaving committed states are enabled.

A.6 Model checking

Verification in UPPAAL is done using model checking. Basically, model checking explores all possible behaviors. A model M is created in a formalism accepted by the model checking tool (e.g. finite state machine), requirements R are specified in temporal logic, and finally all states of M are found and R is checked in each state [4].

In UPPAAL, the specification language for requirements is a subset of timed computation tree logic. Requirements are expressed in terms of formulae that refer to the computation tree of the system. In Figure A.6 an example of a computation tree is given.

UPPAAL's specification language has two types of path and state quantifiers:

E Exists a path

A For all paths

G All states in path ([] in UPPAAL)

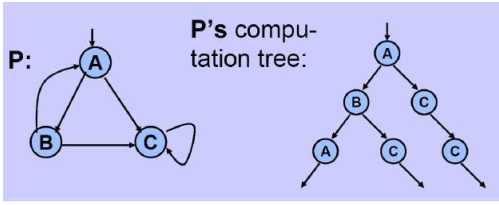


Figure A.6: Example of a computation tree

F Some state in path (<> in UPPAAL)

The standard operators UPPALL allows are:

1. `E<>` `p`: For some path, p holds in some state (p holds sometime)
2. `A[]` `p`: For all paths, p holds in all states (p holds invariantly)
3. `E[]` `p`: For some path, p holds in all states (p holds potentially always)
4. `A<>` `p`: For all paths, p holds in some state (p holds inevitably)

Door example - model checking

If we want to verify that the door cannot be opened more than 100 times, this can be expressed in UPPAAL by the following query:

$$A[]D.count < 101$$

And the response from the UPPAAL model checker is `Property is satisfied`

If we wanted to see whether it was possible to open the door 100 times we could instead check the following query:

$$E <> D.count == 100$$

Again the response from the UPPAAL model checker is `Property is satisfied`. If the option *Diagnostic Trace* is checked, the simulator will then provide a trace for a path where the property holds in some state.

Appendix B

Source code for Java frontend

Task.java

```
public class Task{
    int e, d, o, p, pri, proc;
    public Task(int exetime, int deadline, int offset, int period, int
        priority, int procID){
        e=exetime;
        d=deadline;
        o=offset;
        p=period;
        pri=priority;
        proc=procID;
    }
}
```

Processor.java

```
public class Processor{
    int procID,schedType;
    public Processor(int id, byte sch){
        procID=id;
        schedType=sch;
    }

    public static final byte EDF = 1;
    public static final byte RM = 2;
    public static final byte DM = 3;
    public static final byte FP = 4;
```

```
}
```

Application.java

```
public class Application{
    Task[][] tasks;
    int[][] dependencies;
    public Application(Task[][] ts){
        tasks=ts;
        dependencies = new int[nrTasks()][nrTasks()];
        for(int i=0; i<nrTasks(); i++)
            for(int j=0; j<nrTasks(); j++)
                dependencies[i][j]=0;
    }

    private int nrTasks(){
        int nr=0;
        for(int i=0;i<tasks.length;i++)
        nr+=tasks[i].length;
        return nr;
    }

    public void addDep(int from, int to){
        dependencies[to-1][from-1]=1;
    }
}
```

Platform.java

```
public class Platform{
    Processor[] processors;
    public Platform(Processor[] ps){
        processors=ps;
    }
}
```

MPSoC.java

```
public class MPSoC{
    private static Application application;
    private static Platform platform;
    MPSoC(Application app, Platform pl){
        application = app;
        platform = pl;
    }

    private static String toStringDep(int[][] d){
        String res ="{{";
        for (int i=0;i<d.length;i++){
            for (int j=0;j<d[0].length;j++){
                if(j!=(d[0].length-1))
                    res+=d[i][j]+",";
                else
                    res+=d[i][j];
            }
```

```
            if(i==(d.length-1))
                res+="}}";
            else
                res+="},{";
        }
        return res;
    }

    private static String toStringTasks(Task tas,int l, int g){
        return "Taskc"+(g+1)+"=Task_Control("+(tas.proc)+",_"+(g+1)+"
            ,_"+(l+1)+",_"+tas.e+",_"+tas.d+",_"+tas.o+",_"+tas.p+",_"+
            tas.pri+",_tid["+((tas.proc)-1)+"],_gtid["+((tas.proc)-1)+"
            ],_pri["+((tas.proc)-1)+"],_per["+((tas.proc)-1)+"],_dper["
            +((tas.proc)-1)+"],_nowrun["+((tas.proc)-1)+"],_sact["+((tas
            .proc)-1)+"],_edf["+((tas.proc)-1)+"],_pending1["+((tas.proc
            )-1)+"],tact["+g+"]);_\n"+
            "Taski"+(g+1)+"=Task_Idle("+(tas.proc)+",_"+(g+1)+",_"+(l
                +1)+",_"+tas.e+",_"+tas.d+",_"+tas.o+",_"+tas.p+",_"+tas
                .pri+",_tid["+((tas.proc)-1)+"],_gtid["+((tas.proc)-1)+"
                ],_pri["+((tas.proc)-1)+"],_per["+((tas.proc)-1)+"],_
                dper["+((tas.proc)-1)+"],_nowrun["+((tas.proc)-1)+"],_
                sact["+((tas.proc)-1)+"],_edf["+((tas.proc)-1)+"],_
                pending1["+((tas.proc)-1)+"],tact["+g+"]);_\n"+
            "Taskre"+(g+1)+"=Task_Ready("+(tas.proc)+",_"+(g+1)+",_"+(
                l+1)+",_"+tas.e+",_"+tas.d+",_"+tas.o+",_"+tas.p+",_"+
                tas.pri+",_tid["+((tas.proc)-1)+"],_gtid["+((tas.proc)
                -1)+"],_pri["+((tas.proc)-1)+"],_per["+((tas.proc)-1)+"
                ],_dper["+((tas.proc)-1)+"],_nowrun["+((tas.proc)-1)+"
                ],_sact["+((tas.proc)-1)+"],_edf["+((tas.proc)-1)+"],_
                pending1["+((tas.proc)-1)+"],tact["+g+"]);_\n"+
            "Taskru"+(g+1)+"=Task_Running("+(tas.proc)+",_"+(g+1)+",_"
                +(l+1)+",_"+tas.e+",_"+tas.d+",_"+tas.o+",_"+tas.p+",_"+
                tas.pri+",_tid["+((tas.proc)-1)+"],_gtid["+((tas.proc)
                -1)+"],_pri["+((tas.proc)-1)+"],_per["+((tas.proc)-1)+"
                ],_dper["+((tas.proc)-1)+"],_nowrun["+((tas.proc)-1)+"
                ],_sact["+((tas.proc)-1)+"],_edf["+((tas.proc)-1)+"],_
                pending1["+((tas.proc)-1)+"],tact["+g+"]);_\n";
    }

    private static String typeProc(Processor pr){
        switch(pr.schedType){
        case Processor.EDF:
            return "edf";

        case Processor.RM:
            return "per";

        case Processor.DM:
            return "dper";

        case Processor.FP:
            return "pri";
        }
        return "";
    }

    private static String toStringProc(Processor pr, int i){
        return "Con"+(i+1)+"=Control("+(i+1)+",_sact["+i+"],_ftid["+i+
            "],_tid["+i+"],_gtid["+i+"],_rbtid["+i+"],_running["+i+"],_
            cact["+i+"],_pending1["+i+"]);\n"+
    "Syn"+(i+1)+"=Synchron("+(i+1)+",_ftid["+i+"],_rbtid["+i+"],_cact[
        "+i+"],_ltog["+i+"]);\n"+
    "Sch"+(i+1)+"=SchedMin("+(i+1)+",_tid["+i+"],_"+typeProc(pr)+"["+i
        +"],_running["+i+"],_ftid["+i+"],_sact["+i+"],_cact["+i+"],_ltog
        ["+i+"]);\n";
```

```
}

private static String toStringAll(Task[][]ta, Processor[] pa){
    String rs = "";
    int gbl=0;
    for (int i=0; i<ta.length; i++){
        if (ta[i]!=null){
            rs+=toStringProc(pa[i],i);
            for (int j=0; j<ta[i].length; j++){
                if (ta[i][j]!=null){
                    rs+=toStringTasks(ta[i][j],j,gbl);
                    gbl++;
                }
            }
        }
    }
    return rs;
}

private static int nrOfProc(Processor[] p) {return p.length;}
private static int maxNrTasks(Task[][] t) {
    int mx = 0;
    for(int i=0;i<t.length;i++)
        if(t[i].length>mx)
            mx=t[i].length;
    return mx;
}
private static int totalNrTasks(Task[][] t){
    int cnt=0;
    for (int i=0; i<t.length; i++){
        if (t[i]!=null){
            for(int j=0; j<t[i].length; j++){
                if (t[i][j]!=null)
                    cnt++;
            }
        }
    }
    return cnt;
}

private static String localToGlobal(Task[][] t){
    int coun = 1;
    String res = "{{";
    for(int i=0; i<t.length;i++){
        for(int j=0; j<maxNrTasks(t);j++){
            if (j<t[i].length){
                res+=coun;
                coun++;
            }
            else res+="0";
            if(j!=maxNrTasks(t)-1)
                res+=",␣";
        }
        res+="}";
        if(i!=t.length-1)
            res+=",␣{";
    }
    res+="}";
    return res;
}
private static String toStringSystem(Task[][]ta, Processor[] pa){
    String res = "";
    for(int i=1;i<totalNrTasks(ta)+1;i++){
        res+="Taskc"+i+",␣Taski"+i+",␣Taskre"+i+",␣Taskru"+i+",";
    }
```

```
        for(int i=1;i<nrOfProc(pa);i++){
            res+="Con"+i+",Syn"+i+",Sch"+i+",";
        }
        res+="Con"+nrOfProc(pa)+",Syn"+nrOfProc(pa)+",Sch"+nrOfProc(pa
            );
        return res;
    }

    static String mkXML() {
        return "<?xml version='1.0' encoding='utf-8'?><!DOCTYPE nta
            PUBLIC '-//Uppaal Team//DTD Flat System 1.1//EN' 'http://www
            .it.uu.se/research/group/darts/uppaal/flat-1_1.dtd'><nta><
            declaration>const int M=" +nrOfProc(platform.processors)+"
            ;//The number of processors \n"+
"const int N="+maxNrTasks(application.tasks)+";//The maximum number
    of tasks per Processor\n"+
"const int MN="+totalNrTasks(application.tasks)+";//The total number
    of tasks\n"+
"//symbolic representation of scheduling actions\n"+
"const int REA=0,RUN=1,PRE=2,FIN=3,FIN2=4,RUN2=5,REA2
    =6,FINa=7,FIN2a=8;\n"+
"const int SYN=0,SCH=1;//symbolic representation of internal
    processor synchronizing\n"+
"int[0,MN] ltog[M][N]="+localToGlobal(application.tasks)+";//global
    taskids from locals\n"+
"broadcast chan resch;//broadcast channel for rescheldluling after a
    task has finished\n"+
"chan cact[M][2];//channel array for internal processor actions\n"+
"chan sact[M][4];//channel array for scheduling actions\n"+
"chan tact[MN][9];//channel array for internal task synchronization\n"+
"\n"+
"clock cp[MN];//array of period clocks for the tasks\n"+
"int cr[MN];//array of variables containing the running time for the
    tasks\n"+
"\n"+
"int[0,N] tid[M];//transfer of local taskid from task to controller\n"+
"int[0,MN] gtid[M];//transfer of global taskid from task to controller\
    n"+
"int[0,MN] ftid[M];//taskid of task which has finished, ftid=0 means no
    finished task\n"+
"bool rbtid[M][N];//array of tasks which have become ready\n"+
"bool running[M];//indicating wether a task is currently running on the
    processor\n"+
"bool tasksReady[MN];//array of tasks which are ready for scheduling\n"
    +
"\n"+
"\n"+
"//Criteria usable in scheduling\n"+
"int pri[M][N];//criterion used for user defined priority scheduling\n"
    +
"int per[M][N];//criterion used for rate monotonic scheduling\n"+
"int dper[M][N];//criterion used for deadline monotonic scheduling\n"+
"int edf[M][N];//criterion used for earliest deadline first scheduling\
    n"+
"\n"+
"//array for original dependencies, 1 for dependency, 0 for no
    dependency\n"+
"bool origdep[MN][MN]="+toStringDep(application.dependencies)+";\n"+
"\n"+
"//dynamically updated array for current dependencies\n"+
"bool depend[MN][MN]="+toStringDep(application.dependencies)+";\n"+
"\n"+
"bool wtdep[MN];//array for tasks ready but waiting for dependencies to
    be resolved\n"+
"\n"+
```

```
"//Locking␣mechanisms\n"+
"bool␣nowrun[M];␣//ensuring␣completion␣of␣'runs'␣before␣reacting␣on␣
    ready\n"+
"bool␣pending[MN];␣//ensuring␣global␣wait␣for␣finish␣&amp;␣ready␣before␣
    next␣'run'\n"+
"bool␣pending1[M][N];␣//ensuring␣reaction␣on␣all␣ready␣signals␣available
    ␣locally\n"+
"bool␣h_edf[MN];␣//ensuring␣synchronization␣on␣extra␣state␣in␣ready␣for␣
    edf\n"+
"bool␣h_t[M];␣//ensuring␣completion␣of␣all␣'runs'␣before␣next␣'run'\n"+
"bool␣h_fin[M];␣//ensuring␣reaction␣on␣finish␣before␣ready\n"+
"bool␣l_in[M];␣//ensuring␣completion␣of␣local␣scheduling␣before␣glocal␣
    reschedule\n"+
"bool␣l_out[M];␣//ensuring␣reaction␣on␣reschedule␣before␣next␣'run'\n"+
"bool␣hold[M];␣//ensuring␣3-in-one␣completion␣of␣transitions␣on␣
    Task_Control\n"+
"bool␣h_r[M];␣//ensuring␣reschedule␣before␣next␣'run'\n"+
"\n"+
"//function␣checking␣for␣dependencies␣for␣task␣t\n"+
"bool␣dep(int␣t)\n"+
"{\n"+
"␣␣for␣(ini␣:␣int[0,MN-1])\n"+
"␣␣{␣␣␣␣if␣(depend[t][ini])\n"+
"␣␣␣␣{\n"+
"␣␣␣␣␣␣return␣true;\n"+
"␣␣␣␣}\n"+
"␣␣}\n"+
"␣␣return␣false;\n"+
"}\n"+
"//function␣updating␣dependencies␣when␣task␣t␣has␣finished\n"+
"void␣opdDep(int␣t)\n"+
"{\n"+
"␣␣for␣(ini␣:␣int[0,MN-1])\n"+
"␣␣{\n"+
"␣␣␣␣depend[ini][t]=false;\n"+
"␣␣}\n"+
"}\n"+
"//function␣setting␣the␣original␣dependency␣values␣for␣task␣t\n"+
"void␣setOrigDep(int␣t)\n"+
"{\n"+
"␣␣for␣(ini␣:␣int[0,MN-1])\n"+
"␣␣{\n"+
"␣␣␣␣depend[t][ini]=origdep[t][ini];\n"+
"␣␣}\n"+
"}\n"+
"//function␣checking␣for␣existance␣of␣boolean␣value␣in␣array␣of␣size␣M\n
    "+
"bool␣locked(bool␣la[M])\n"+
"{\n"+
"␣␣for␣(ini␣:␣int[0,M-1])\n"+
"␣␣{␣␣␣␣if␣(la[ini])\n"+
"␣␣␣␣{\n"+
"␣␣␣␣␣␣return␣true;\n"+
"␣␣␣␣}\n"+
"␣␣}\n"+
"␣␣return␣false;\n"+
"}\n"+
"//function␣checking␣for␣existance␣of␣boolean␣value␣in␣array␣of␣size␣N\n
    "+
"bool␣pend(bool␣pen[N])\n"+
"{\n"+
"␣␣bool␣b␣=␣false;\n"+
"␣␣for␣(ini␣:␣int[0,N-1])␣\n"+
"␣␣{\n"+
"␣␣␣␣if␣(pen[ini]␣==␣true)\n"+
```

```
" ␣␣␣␣␣␣{\n"+
" ␣␣␣␣␣␣␣␣return ␣true;␣␣\n"+
" ␣␣␣␣␣␣}\n"+
" ␣␣}\n"+
" ␣␣return ␣false;\n"+
"}\n"+
"//function ␣checking ␣for ␣existance ␣of ␣boolean ␣value ␣in ␣array ␣of ␣size ␣MN\
    n"+
"bool ␣pendMN(bool ␣pen[MN])\n"+
"{\n"+
" ␣␣bool ␣b␣=␣false;\n"+
" ␣␣for ␣(ini ␣: ␣int[0,MN-1]) ␣\n"+
" ␣␣{\n"+
" ␣␣␣␣if ␣(pen[ini]␣==␣true)\n"+
" ␣␣␣␣␣␣{\n"+
" ␣␣␣␣␣␣␣␣return ␣true;␣␣\n"+
" ␣␣␣␣␣␣}\n"+
" ␣␣}\n"+
" ␣␣return ␣false;\n"+
"}\n"+
"</declaration><template><name>Task␣Ready</name><parameter>const ␣int␣
    schnr ,␣const␣int␣gtasknr ,␣const␣int␣tasknr ,␣const␣int␣e ,␣const␣int␣
    dead ,␣const␣int␣offset ,␣const␣int␣ptime ,␣int␣prior ,␣int[0,N]␣&amp;
    tid ,␣int[0,MN]␣&amp;gtid ,␣int␣&amp;pri[N] ,␣int␣&amp;per[N] ,␣int␣&amp
    ;dper[N] ,␣bool␣&amp;nowrun ,chan␣&amp;sact[4] ,␣int␣&amp;edf[N] ,␣bool
    ␣&amp;pend1[N] ,␣chan␣&amp;tact[9]</parameter><declaration>clock␣x;</
    declaration><location␣id=\"id0\"␣x=\"40\"␣y=\"32\"></location><
    location␣id=\"id1\"␣x=\"32\"␣y=\"-128\"><label␣kind=\"invariant\"␣x
    =\"22\"␣y=\"-113\">x&lt;=1</label></location><location␣id=\"id2\"␣x
    =\"-160\"␣y=\"-128\"><label␣kind=\"invariant\"␣x=\"-248\"␣y
    =\"-112\">cp[gtasknr-1]&lt;=dead-cr[gtasknr-1]+1\n"+
"&amp;&amp;␣x&lt;1</label></location><init␣ref=\"id0\"/><transition><
    source␣ref=\"id2\"/><target␣ref=\"id0\"/><label␣kind=\"
    synchronisation\"␣x=\"-160\"␣y=\"8\">tact[RUN]!</label><nail␣x
    =\"-160\"␣y=\"32\"/></transition><transition><source␣ref=\"id1\"/><
    target␣ref=\"id2\"/><label␣kind=\"guard\"␣x=\"-112\"␣y=\"-264\">x
    ==1</label><label␣kind=\"assignment\"␣x=\"-112\"␣y=\"-248\">edf[
    tasknr-1]--,␣\n"+
"x=0,\n"+
"h␣edf[gtasknr-1]=false</label><nail␣x=\"-72\"␣y=\"-200\"/></transition><
    transition><source␣ref=\"id2\"/><target␣ref=\"id1\"/><label␣kind=\"guard
    \"␣x=\"-120\"␣y=\"-144\">x&gt;0␣&amp;&amp;␣x&lt;1</label><label␣kind=\"
    assignment\"␣x=\"-120\"␣y=\"-128\">h␣edf[gtasknr-1]=true</label></transition
    ><transition><source␣ref=\"id0\"/><target␣ref=\"id2\"/><label␣kind=\"
    synchronisation\"␣x=\"-64\"␣y=\"-40\">tact[REA2]?</label><label␣kind=\"
    assignment\"␣x=\"-64\"␣y=\"-24\">x=0</label></transition></template><
    template><name>Task␣Control</name><parameter>const␣int␣schnr ,␣const␣int
    ␣gtasknr ,␣const␣int␣tasknr ,␣const␣int␣e ,␣const␣int␣dead ,␣const␣int␣offset
    ,␣const␣int␣ptime ,␣int␣prior ,␣int[0,N]␣&amp;tid ,␣int[0,MN]␣&amp;gtid ,␣
    int␣&amp;pri[N] ,␣int␣&amp;per[N] ,␣int␣&amp;dper[N] ,␣bool␣&amp;nowrun ,
    chan␣&amp;sact[4] ,␣int␣&amp;edf[N] ,␣bool␣&amp;pend1[N] ,␣chan␣&amp;tact
    [9]</parameter><location␣id=\"id3\"␣x=\"40\"␣y=\"0\"><committed/></location
    ><location␣id=\"id4\"␣x=\"-216\"␣y=\"8\"><committed/></location><location
    ␣id=\"id5\"␣x=\"256\"␣y=\"-128\"><committed/></location><location␣id=\"
    id6\"␣x=\"256\"␣y=\"32\"><committed/></location><location␣id=\"id7\"␣
    x=\"-88\"␣y=\"-112\"><committed/></location><location␣id=\"id8\"␣x=\"-48\"␣
    y=\"104\"><committed/></location><location␣id=\"id9\"␣x=\"128\"␣y=\"32\"><
    committed/></location><location␣id=\"id10\"␣x=\"128\"␣y=\"-128\"><committed
    /></location><location␣id=\"id11\"␣x=\"-32\"␣y=\"-256\"><committed/></
    location><location␣id=\"id12\"␣x=\"-256\"␣y=\"-256\"><committed/></location
    ><location␣id=\"id13\"␣x=\"128\"␣y=\"160\"><name␣x=\"104\"␣y=\"176\">
    Running</name></location><location␣id=\"id14\"␣x=\"128\"␣y=\"-256\"><
    name␣x=\"118\"␣y=\"-286\">Ready</name></location><location␣id=\"id15\"␣
    x=\"-416\"␣y=\"-256\"><name␣x=\"-426\"␣y=\"-286\">Idle</name></location
    ><init␣ref=\"id15\"/><transition><source␣ref=\"id13\"/><target␣ref=\"
```

```
      id3\"/><label kind=\"synchronisation\" x=\"0\" y=\"24\">tact[FINa]?</
      label><label kind=\"assignment\" x=\"0\" y=\"40\">hold[schnr-1]=true</
      label></transition><transition><source ref=\"id4\"/><target ref=\"id15
      \"/><label kind=\"synchronisation\" x=\"-304\" y=\"-48\">tact[FIN2]!</
      label><label kind=\"assignment\" x=\"-304\" y=\"-32\">hold[schnr-1]=false
      </label></transition><transition><source ref=\"id8\"/><target ref=\"id4
      \"/><label kind=\"synchronisation\" x=\"-200\" y=\"24\">sact[FIN]!</label
      ><label kind=\"assignment\" x=\"-200\" y=\"40\">pending[gtasknr-1]=false
      ,\n"+
"tid=tasknr,\n"+
"gtid=gtasknr,\n"+
"edf[tasknr-1]=0,\n"+
"h_fin[schnr-1]=false</label></transition><transition><source ref=\"id7
      \"/><target ref=\"id15\"/><label kind=\"synchronisation\" x
      =\"-176\" y=\"-160\">tact[FIN2a]!</label><label kind=\"assignment\"
      x=\"-176\" y=\"-144\">hold[schnr-1]=false</label></transition>
      <transition><source ref=\"id3\"/><target ref=\"id7\"/><label kind=\"
      synchronisation\" x=\"-64\" y=\"-104\">sact[FIN]!</label><label kind
      =\"assignment\" x=\"-64\" y=\"-88\">tid=tasknr,\n"+
"gtid=gtasknr,\n"+
"edf[tasknr-1]=0,\n"+
"h_fin[schnr-1]=false,\n"+
"pend1[tasknr-1]=true</label></transition><transition><source ref=\"id5\"/><
      target ref=\"id14\"/><label kind=\"synchronisation\" x=\"176\" y=\"-224\">
      tact[REA2]!</label><label kind=\"assignment\" x=\"176\" y=\"-208\">hold
      [schnr-1]=false</label></transition><transition><source ref=\"id6\"/><
      target ref=\"id5\"/><label kind=\"synchronisation\" x=\"224\" y=\"-56\">
      tact[PRE]?</label></transition><transition><source ref=\"id9\"/><target
      ref=\"id13\"/><label kind=\"synchronisation\" x=\"96\" y=\"64\">tact
      [RUN2]!</label><label kind=\"assignment\" x=\"96\" y=\"80\">hold[schnr
      -1]=false</label></transition><transition><source ref=\"id10\"/><target
      ref=\"id9\"/><label kind=\"synchronisation\" x=\"104\" y=\"-56\">tact
      [RUN]?</label></transition><transition><source ref=\"id11\"/><target
      ref=\"id14\"/><label kind=\"synchronisation\" x=\"16\" y=\"-256\">tact
      [REA2]!</label><label kind=\"assignment\" x=\"-12\" y=\"-240\">hold[schnr
      -1]=false</label></transition><transition><source ref=\"id12\"/><target
      ref=\"id11\"/><label kind=\"synchronisation\" x=\"-176\" y=\"-256\">
      tact[REA]?</label></transition><transition><source ref=\"id13\"/><target
      ref=\"id8\"/><label kind=\"synchronisation\" x=\"0\" y=\"120\">tact[
      FIN]?</label><label kind=\"assignment\" x=\"-8\" y=\"136\">hold[schnr
      -1]=true</label></transition><transition><source ref=\"id13\"/><target
      ref=\"id6\"/><label kind=\"guard\" x=\"184\" y=\"112\">cr[gtasknr-1]&
      gt;0</label><label kind=\"synchronisation\" x=\"184\" y=\"128\">sact[
      PRE]?</label><label kind=\"assignment\" x=\"184\" y=\"144\">hold[schnr
      -1]=true</label></transition><transition><source ref=\"id14\"/><target
      ref=\"id10\"/><label kind=\"guard\" x=\"80\" y=\"-208\">tid==tasknr</
      label><label kind=\"synchronisation\" x=\"80\" y=\"-192\">sact[RUN]?</
      label><label kind=\"assignment\" x=\"80\" y=\"-176\">hold[schnr-1]=true
      </label></transition><transition><source ref=\"id15\"/><target ref=\"
      id12\"/><label kind=\"guard\" x=\"-392\" y=\"-336\">cp[gtasknr-1]==ptime
      &amp;&amp;\n"+
"!nowrun&amp;&amp;\n"+
"!locked(h_fin)</label><label kind=\"synchronisation\" x=\"-392\" y
      =\"-288\">sact[REA]!</label><label kind=\"assignment\" x=\"-392\" y
      =\"-272\">cp[gtasknr-1]=0,\n"+
"tid=tasknr,\n"+
"gtid=gtasknr,\n"+
"pending[gtasknr-1]=false,\n"+
"pend1[tasknr-1]=false,\n"+
"edf[tasknr-1]=dead,\n"+
"setOrigDep(gtasknr-1),\n"+
"hold[schnr-1]=true,\n"+
"cr[gtasknr-1]=e</label></transition></template><template><name>SchedMin
      </name><parameter>const int schednr, int[0,N]&amp;tid, int&amp;cri
      [N], bool&amp;running, int[0,MN]&amp;ftid, chan&amp;sact[4], chan
```

```
␣&amp;cact[2],␣int[0,MN]␣&amp;ltog[N]</parameter><declaration>int[0,
    N]␣it;␣//iteration␣variable\n"+
"int␣lcri;␣//variable␣used␣to␣hold␣the␣criterion␣of␣the␣task␣currently␣
    chosen\n"+
"int[0,N]␣ltid;␣//variable␣used␣to␣hold␣the␣id␣of␣the␣task␣currently␣
    running\n"+
"int[0,N]␣curtid;␣//variable␣used␣to␣hold␣the␣id␣of␣the␣task␣currently␣
    chosen\n"+
"int[0,MN]␣li;␣//variable␣used␣to␣hold␣global␣id␣for␣tasks\n"+
"</declaration><location␣id=\"id16\"␣x=\"40\"␣y=\"-208\"><committed/></location
    ><location␣id=\"id17\"␣x=\"192\"␣y=\"8\"><committed/></location><location
    ␣id=\"id18\"␣x=\"-192\"␣y=\"-208\"><committed/></location><location␣id
    =\"id19\"␣x=\"-360\"␣y=\"-64\"><committed/></location><location␣id=\"
    id20\"␣x=\"-280\"␣y=\"-64\"></location><location␣id=\"id21\"␣x=\"-192\"␣
    y=\"-64\"><committed/></location><location␣id=\"id22\"␣x=\"192\"␣y=\"-120\"><
    committed/></location><location␣id=\"id23\"␣x=\"-320\"␣y=\"8\"><committed
    /></location><location␣id=\"id24\"␣x=\"40\"␣y=\"8\"><committed/></location
    ><location␣id=\"id25\"␣x=\"40\"␣y=\"-120\"><committed/></location><init
    ␣ref=\"id20\"/><transition><source␣ref=\"id16\"/><target␣ref=\"id24\"/><
    label␣kind=\"guard\"␣x=\"-56\"␣y=\"-112\">it==N</label><nail␣x=\"-24\"␣
    y=\"-104\"/></transition><transition><source␣ref=\"id16\"/><target␣ref
    =\"id25\"/><label␣kind=\"guard\"␣x=\"16\"␣y=\"-192\">it&lt;N</label><
    label␣kind=\"assignment\"␣x=\"-16\"␣y=\"-176\">li=ltog[it]</label></transition
    ><transition><source␣ref=\"id17\"/><target␣ref=\"id24\"/><label␣kind=\"
    guard\"␣x=\"64\"␣y=\"56\">tasksReady[li-1]\n"+
"&amp;amp;&amp;␣cri[it]␣&lt;␣lcri</label><label␣kind=\"assignment\"␣x
    =\"64\"␣y=\"88\">curtid=it+1,\n"+
"lcri=cri[it]</label><nail␣x=\"112\"␣y=\"64\"/></transition><transition
    ><source␣ref=\"id17\"/><target␣ref=\"id24\"/><label␣kind=\"guard\"␣x
    =\"64\"␣y=\"-8\">cri[it]␣&gt;=lcri␣||\n"+
"!tasksReady[li-1]</label></transition><transition><source␣ref=\"id22\"/><
    target␣ref=\"id17\"/><label␣kind=\"guard\"␣x=\"176\"␣y=\"-40\">it==N-1</
    label></transition><transition><source␣ref=\"id18\"/><target␣ref=\"id21
    \"/><label␣kind=\"guard\"␣x=\"-160\"␣y=\"-152\">!tasksReady[li-1]</label
    ><label␣kind=\"assignment\"␣x=\"-144\"␣y=\"-136\">it++</label><nail␣x
    =\"-128\"␣y=\"-144\"/></transition><transition><source␣ref=\"id21\"/><
    target␣ref=\"id18\"/><label␣kind=\"guard\"␣x=\"-288\"␣y=\"-184\">it&lt
    ;N␣&amp;&amp;␣ltog[it]␣!=␣0</label><label␣kind=\"assignment\"␣x=\"-256\"␣
    y=\"-168\">li=ltog[it]</label></transition><transition><source␣ref=\"
    id21\"/><target␣ref=\"id19\"/><label␣kind=\"guard\"␣x=\"-344\"␣y=\"-152\">
    it==N␣||␣ltog[it]==0</label><nail␣x=\"-288\"␣y=\"-136\"/></transition
    ><transition><source␣ref=\"id19\"/><target␣ref=\"id20\"/><label␣kind=\"
    guard\"␣x=\"-376\"␣y=\"-104\">!hold[schednr-1]</label><label␣kind=\"synchronisation
    \"␣x=\"-352\"␣y=\"-88\">cact[SCH]!</label></transition><transition><source
    ␣ref=\"id20\"/><target␣ref=\"id21\"/><label␣kind=\"synchronisation\"␣
    x=\"-264\"␣y=\"-96\">cact[SCH]?</label><label␣kind=\"assignment\"␣x=\"-264\"␣
    y=\"-80\">it=0</label></transition><transition><source␣ref=\"id25\"/><
    target␣ref=\"id22\"/><label␣kind=\"guard\"␣x=\"104\"␣y=\"-136\">li␣!=␣0</
    label></transition><transition><source␣ref=\"id24\"/><target␣ref=\"id23
    \"/><label␣kind=\"guard\"␣x=\"-264\"␣y=\"72\">!running</label><label␣
    kind=\"assignment\"␣x=\"-264\"␣y=\"88\">ltid=curtid,\n"+
"tid=curtid</label><nail␣x=\"-112\"␣y=\"88\"/><nail␣x=\"-320\"␣y
    =\"88\"/></transition><transition><source␣ref=\"id23\"/><target␣ref
    =\"id19\"/><label␣kind=\"guard\"␣x=\"-344\"␣y=\"-56\">!hold[schednr
    -1]</label><label␣kind=\"synchronisation\"␣x=\"-344\"␣y=\"-40\">sact
    [RUN]!</label><label␣kind=\"assignment\"␣x=\"-344\"␣y=\"-24\">
    running␣=␣true</label><nail␣x=\"-320\"␣y=\"-32\"/></transition><
    transition><source␣ref=\"id24\"/><target␣ref=\"id23\"/><label␣kind
    =\"guard\"␣x=\"-184\"␣y=\"-24\">ltid!=curtid␣&amp;&amp;\n"+
"running</label><label␣kind=\"synchronisation\"␣x=\"-184\"␣y=\"8\">sact[
    PRE]!</label><label␣kind=\"assignment\"␣x=\"-184\"␣y=\"24\">tid=
    curtid,\n"+
"ltid=curtid</label></transition><transition><source␣ref=\"id24\"/><
    target␣ref=\"id19\"/><label␣kind=\"guard\"␣x=\"-160\"␣y=\"104\">ltid
    ==curtid␣&amp;&amp;\n"+
```

```
"running</label><nail x=\"-80\" y=\"136\"/><nail x=\"-360\" y
    =\"136\"/></transition><transition><source ref=\"id25\"/><target ref
    =\"id24\"/><label kind=\"guard\" x=\"8\" y=\"-80\">li==0</label></
    transition><transition><source ref=\"id22\"/><target ref=\"id25\"/><
    label kind=\"guard\" x=\"56\" y=\"-88\">(cri[it]&gt;=lcri||\n"+
"!tasksReady[li-1])&amp;&amp;\n"+
"it&lt;N-1</label><label kind=\"assignment\" x=\"56\" y=\"-40\">it++,\n"
    +
"li=ltog[it]</label><nail x=\"120\" y=\"-80\"/></transition><transition
    ><source ref=\"id22\"/><target ref=\"id25\"/><label kind=\"guard\" x
    =\"80\" y=\"-248\">tasksReady[li-1]\n"+
"&amp;&amp; cri[it]&lt;lcri\n"+
"&amp;&amp; it&lt;N-1</label><label kind=\"assignment\" x=\"80\" y
    =\"-208\">curtid=it+1,\n"+
"lcri=cri[it],\n"+
"it++,li=ltog[it]</label><nail x=\"120\" y=\"-160\"/></transition>
    <transition><source ref=\"id18\"/><target ref=\"id16\"/><label kind
    =\"guard\" x=\"-128\" y=\"-224\">tasksReady[li-1]</label><label kind
    =\"assignment\" x=\"-128\" y=\"-208\">curtid=it+1,\n"+
"lcri=cri[it],\n"+
"it++</label></transition></template><template><name>Control</name><parameter
    >const int schnr,chan &amp;sact[4],int[0,MN] &amp;ftid,int[0,N] &amp
    ;tid,int[0,MN] &amp;gtid,bool &amp;rbtid[N],bool &amp;running,chan
    &amp;cact[2],bool &amp;pend1[N]</parameter><location id=\"id26\" x=\"-88\"
    y=\"-192\"><committed/></location><location id=\"id27\" x=\"-96\" y=\"8\"><
    urgent/></location><location id=\"id28\" x=\"-112\" y=\"48\"><committed
    /></location><location id=\"id29\" x=\"96\" y=\"48\"><committed/></location
    ><location id=\"id30\" x=\"96\" y=\"-112\"><urgent/></location><location
    id=\"id31\" x=\"-88\" y=\"-112\"><committed/></location><location id
    =\"id32\" x=\"-304\" y=\"-112\"></location><init ref=\"id32\"/><transition
    ><source ref=\"id26\"/><target ref=\"id30\"/><label kind=\"guard\" x=\"-8\"
    y=\"-192\">!hold[schnr-1]</label><label kind=\"synchronisation\" x=\"-8\"
    y=\"-176\">cact[SYN]!</label></transition><transition><source ref=\"id31
    \"/><target ref=\"id31\"/><label kind=\"guard\" x=\"-120\" y=\"-88\">!
    hold[schnr-1]</label><label kind=\"synchronisation\" x=\"-120\" y=\"-72\">
    sact[REA]?</label><label kind=\"assignment\" x=\"-120\" y=\"-56\">rbtid
    [tid-1]=true,\n"+
"pend1[tid-1]=false</label><nail x=\"-104\" y=\"-72\"/><nail x=\"-72\" y
    =\"-72\"/></transition><transition><source ref=\"id27\"/><target ref
    =\"id29\"/><label kind=\"guard\" x=\"-56\" y=\"-24\">!locked(l_in)</
    label><label kind=\"assignment\" x=\"-56\" y=\"-8\">l_out[schnr-1]=
    false</label><nail x=\"64\" y=\"8\"/></transition><transition>
    <source ref=\"id32\"/><target ref=\"id27\"/><label kind=\"
    synchronisation\" x=\"-248\" y=\"-24\">resch?</label><label kind=\"
    assignment\" x=\"-248\" y=\"-8\">l_out[schnr-1]=true,\n"+
"ftid=0</label><nail x=\"-272\" y=\"8\"/></transition><transition>
    <source ref=\"id28\"/><target ref=\"id32\"/><label kind=\"
    synchronisation\" x=\"-240\" y=\"48\">cact[SCH]?</label><nail x
    =\"-304\" y=\"48\"/></transition><transition><source ref=\"id29\"/><
    target ref=\"id28\"/><label kind=\"synchronisation\" x=\"-40\" y
    =\"48\">cact[SCH]!</label></transition><transition><source ref=\"
    id30\"/><target ref=\"id29\"/><label kind=\"synchronisation\" x
    =\"96\" y=\"-48\">cact[SYN]?</label></transition><transition><source
    ref=\"id31\"/><target ref=\"id30\"/><label kind=\"guard\" x
    =\"-64\" y=\"-144\">!pend(pend1)&amp;&amp;\n"+
"!hold[schnr-1]</label><label kind=\"synchronisation\" x=\"-64\" y
    =\"-112\">cact[SYN]!</label></transition><transition><source ref=\"
    id32\"/><target ref=\"id31\"/><label kind=\"synchronisation\" x
    =\"-264\" y=\"-128\">sact[REA]?</label><label kind=\"assignment\" x
    =\"-264\" y=\"-112\">rbtid[tid-1]=true,\n"+
"ftid=0,\n"+
"pend1[tid-1]=false</label></transition><transition><source ref=\"id32
    \"/><target ref=\"id26\"/><label kind=\"synchronisation\" x
    =\"-240\" y=\"-224\">sact[FIN]?</label><label kind=\"assignment\" x
    =\"-240\" y=\"-208\">ftid=gtid,\n"+
```

```
"running=false </label></transition></template><template><name>Synchron</
    name><parameter>const int schnr, int [0,MN]&amp;ftid, bool&amp;
    rbtid[N], chan&amp;cact[2], int [0,MN]&amp;ltog[N]</parameter><
    declaration>int [0,MN] i;// iteration variable\n"+
"int [0,MN] li;// variable used to hold global id for tasks\n"+
"bool depCh;// flag used if a dependency has been changed\n"+
"</declaration><location id=\"id33\" x=\"-224\" y=\"-40\"><committed/></location
    ><location id=\"id34\" x=\"-224\" y=\"-160\"><urgent/></location><location
    id=\"id35\" x=\"136\" y=\"96\"><committed/></location><location id=\"
    id36\" x=\"136\" y=\"-40\"><committed/></location><location id=\"id37
    \" x=\"-24\" y=\"-256\"><committed/></location><location id=\"id38\"
    x=\"-24\" y=\"-160\"><committed/></location><location id=\"id39\" x=\"-24\"
    y=\"-40\"><committed/></location><location id=\"id40\" x=\"-152\" y=\"-40\"></
    location><init ref=\"id40\"/><transition><source ref=\"id33\"/><target
    ref=\"id40\"/><label kind=\"synchronisation\" x=\"-224\" y=\"-32\">cact
    [SYN]!</label></transition><transition><source ref=\"id38\"/><target
    ref=\"id40\"/><label kind=\"guard\" x=\"-120\" y=\"-136\">!depCh&amp
    ;&amp;\n"+
"i==MN</label><label kind=\"synchronisation\" x=\"-120\" y=\"-104\">cact
    [SYN]!</label></transition><transition><source ref=\"id34\"/><target
    ref=\"id33\"/><label kind=\"guard\" x=\"-248\" y=\"-144\">!locked(
    h_fin)</label><label kind=\"synchronisation\" x=\"-248\" y=\"-128\">
    resch!</label><label kind=\"assignment\" x=\"-248\" y=\"-112\">depCh
    =false,\n"+
"h_r[schnr-1]=false </label></transition><transition><source ref=\"id38
    \"/><target ref=\"id34\"/><label kind=\"guard\" x=\"-168\" y
    =\"-208\">i==MN&amp;&amp;\n"+
"depCh</label><label kind=\"assignment\" x=\"-168\" y=\"-176\">h_r[schnr
    -1]=true,\n"+
"i=0</label></transition><transition><source ref=\"id36\"/><target ref
    =\"id36\"/><label kind=\"guard\" x=\"184\" y=\"-64\">i&lt;N&amp;&
    amp;\n"+
"!rbtid[i]</label><label kind=\"assignment\" x=\"184\" y=\"-32\">i++</
    label><nail x=\"184\" y=\"-16\"/><nail x=\"184\" y=\"-56\"/></
    transition><transition><source ref=\"id36\"/><target ref=\"id35\"/><
    label kind=\"guard\" x=\"104\" y=\"-8\">i&lt;N&amp;&amp;\n"+
"rbtid[i]</label><label kind=\"assignment\" x=\"104\" y=\"24\">li=ltog[i
    ],\n"+
"rbtid[i]=false </label></transition><transition><source ref=\"id38\"/><
    target ref=\"id38\"/><label kind=\"guard\" x=\"32\" y=\"-192\">i&lt;
    MN&amp;&amp;\n"+
"!wtdep[i]</label><label kind=\"assignment\" x=\"32\" y=\"-160\">i++</
    label><nail x=\"32\" y=\"-184\"/><nail x=\"32\" y=\"-144\"/></
    transition><transition><source ref=\"id36\"/><target ref=\"id40\"/><
    label kind=\"guard\" x=\"-72\" y=\"-16\">i==N</label><label kind=\"
    synchronisation\" x=\"-72\" y=\"0\">cact[SYN]!</label><nail x
    =\"-16\" y=\"-8\"/></transition><transition><source ref=\"id35\"/><
    target ref=\"id36\"/><label kind=\"guard\" x=\"16\" y=\"16\">!dep(li
    -1)</label><label kind=\"assignment\" x=\"-40\" y=\"32\">tasksReady[
    li-1]=true,\n"+
"i++</label><nail x=\"72\" y=\"32\"/></transition><transition><source
    ref=\"id35\"/><target ref=\"id36\"/><label kind=\"guard\" x=\"200\"
    y=\"8\">dep(li-1)</label><label kind=\"assignment\" x=\"200\" y
    =\"24\">wtdep[li-1]=true,\n"+
"i++</label><nail x=\"200\" y=\"32\"/></transition><transition><source
    ref=\"id37\"/><target ref=\"id38\"/><label kind=\"guard\" x=\"8\" y
    =\"-264\">!dep(i)</label><label kind=\"assignment\" x=\"8\" y
    =\"-248\">tasksReady[i]=true,\n"+
"wtdep[i]=false,i++,\n"+
"depCh=true</label><nail x=\"32\" y=\"-216\"/></transition><transition>
    source ref=\"id37\"/><target ref=\"id38\"/><label kind=\"guard\" x
    =\"-96\" y=\"-240\">dep(i)</label><label kind=\"assignment\" x
    =\"-96\" y=\"-224\">i++</label><nail x=\"-80\" y=\"-208\"/></
    transition><transition><source ref=\"id38\"/><target ref=\"id37\"/><
    label kind=\"guard\" x=\"-56\" y=\"-232\">i&lt;MN&amp;&amp;\n"+
```

```
"wtdep[i]</label></transition><transition><source␣ref=\"id39\"/><target␣
    ref=\"id36\"/><label␣kind=\"guard\"␣x=\"24\"␣y=\"-56\">ftid==0</
    label></transition><transition><source␣ref=\"id39\"/><target␣ref=\"
    id38\"/><label␣kind=\"guard\"␣x=\"-40\"␣y=\"-128\">ftid!=0</label><
    label␣kind=\"assignment\"␣x=\"-40\"␣y=\"-112\">tasksReady[ftid-1]=
    false,\n"+
"opdDep(ftid-1)</label></transition><transition><source␣ref=\"id40\"/><
    target␣ref=\"id39\"/><label␣kind=\"synchronisation\"␣x=\"-120\"␣y
    =\"-72\">cact[SYN]?</label><label␣kind=\"assignment\"␣x=\"-112\"␣y
    =\"-56\">i=0</label></transition></template>"+
"<template><name>Task_Idle</name><parameter>const␣int␣schnr,␣const␣int␣gtasknr
    ,␣const␣int␣tasknr,␣const␣int␣e,␣const␣int␣dead,␣const␣int␣offset,␣const
    ␣int␣ptime,␣int␣prior,␣int[0,N]␣&amp;tid,␣int[0,MN]␣&amp;gtid,␣int␣&amp
    ;pri[N],␣int␣&amp;per[N],␣int␣&amp;dper[N],␣bool␣&amp;nowrun,chan␣&amp
    ;sact[4],␣int␣&amp;edf[N],␣bool␣&amp;pend1[N],␣chan␣&amp;tact[9]</parameter
    ><declaration>clock␣x;</declaration><location␣id=\"id41\"␣x=\"152\"␣y
    =\"-224\"><name␣x=\"142\"␣y=\"-254\">Wait</name></location><location␣
    id=\"id42\"␣x=\"-336\"␣y=\"-64\"><name␣x=\"-352\"␣y=\"-96\">Offset</name
    ><label␣kind=\"invariant\"␣x=\"-384\"␣y=\"-48\">x&lt;=offset-1</label
    ></location><location␣id=\"id43\"␣x=\"-136\"␣y=\"-224\"><name␣x=\"-144\"␣
    y=\"-264\">Idle</name><label␣kind=\"invariant\"␣x=\"-168\"␣y=\"-208\">
    cp[gtasknr-1]&lt;=ptime</label></location><location␣id=\"id44\"␣x=\"-136\"␣
    y=\"-64\"><name␣x=\"-152\"␣y=\"-96\">IdleWait</name><label␣kind=\"invariant
    \"␣x=\"-160\"␣y=\"-56\">cp[gtasknr-1]&lt;ptime</label></location><location
    ␣id=\"id45\"␣x=\"-336\"␣y=\"-224\"><name␣x=\"-352\"␣y=\"-256\">Start</
    name><committed/></location><init␣ref=\"id45\"/><transition><source␣ref
    =\"id45\"/><target␣ref=\"id43\"/><label␣kind=\"guard\"␣x=\"-304\"␣y=\"-336\">
    offset==0</label><label␣kind=\"assignment\"␣x=\"-304\"␣y=\"-320\">cp[
    gtasknr-1]=ptime,\n"+
"pri[tasknr-1]=prior,␣\n"+
"per[tasknr-1]=ptime,␣\n"+
"dper[tasknr-1]=dead,\n"+
"pending[gtasknr-1]=true,\n"+
"pend1[tasknr-1]=true</label></transition><transition><source␣ref=\"id41\"/><
    target␣ref=\"id44\"/><label␣kind=\"synchronisation\"␣x=\"0\"␣y=\"-64\">
    tact[FIN2]?</label><nail␣x=\"152\"␣y=\"-64\"/></transition><transition
    ><source␣ref=\"id41\"/><target␣ref=\"id43\"/><label␣kind=\"synchronisation
    \"␣x=\"-32\"␣y=\"-336\">tact[FIN2a]?</label><nail␣x=\"152\"␣y=\"-320\"/><
    nail␣x=\"-136\"␣y=\"-320\"/></transition><transition><source␣ref=\"id43
    \"/><target␣ref=\"id41\"/><label␣kind=\"synchronisation\"␣x=\"-24\"␣y
    =\"-240\">tact[REA]!</label></transition><transition><source␣ref=\"id44
    \"/><target␣ref=\"id43\"/><label␣kind=\"guard\"␣x=\"-136\"␣y=\"-176\">
    cp[gtasknr-1]&gt;ptime-1</label><label␣kind=\"assignment\"␣x=\"-136\"␣
    y=\"-160\">pending[gtasknr-1]=true,\n"+
"pend1[tasknr-1]=true</label></transition><transition><source␣ref=\"id42
    \"/><target␣ref=\"id44\"/><label␣kind=\"guard\"␣x=\"-304\"␣y
    =\"-80\">x==offset-1</label><label␣kind=\"assignment\"␣x=\"-304\"␣y
    =\"-64\">cp[gtasknr-1]=ptime-1</label></transition><transition><
    source␣ref=\"id45\"/><target␣ref=\"id42\"/><label␣kind=\"guard\"␣x
    =\"-328\"␣y=\"-184\">offset&gt;0</label><label␣kind=\"assignment\"␣x
    =\"-328\"␣y=\"-168\">x=0,\n"+
"pri[tasknr-1]=prior,␣\n"+
"per[tasknr-1]=ptime,␣\n"+
"dper[tasknr-1]=dead</label></transition></template>"+
"<template><name>Task_Running</name><parameter>const␣int␣schnr,␣const␣int
    ␣gtasknr,␣const␣int␣tasknr,␣const␣int␣e,␣const␣int␣dead,␣const␣int␣offset
    ,␣const␣int␣ptime,␣int␣prior,␣int[0,N]␣&amp;tid,␣int[0,MN]␣&amp;gtid,␣
    int␣&amp;pri[N],␣int␣&amp;per[N],␣int␣&amp;dper[N],␣bool␣&amp;nowrun,
    chan␣&amp;sact[4],␣int␣&amp;edf[N],␣bool␣&amp;pend1[N],␣chan␣&amp;tact
    [9]</parameter><declaration>clock␣x;</declaration><location␣id=\"id46
    \"␣x=\"-64\"␣y=\"-160\"><name␣x=\"-74\"␣y=\"-190\">Wait</name></location
    ><location␣id=\"id47\"␣x=\"-64\"␣y=\"160\"><name␣x=\"-74\"␣y=\"130\">
    Running4</name><committed/></location><location␣id=\"id48\"␣x=\"160\"␣
    y=\"160\"><name␣x=\"150\"␣y=\"130\">Running3</name><label␣kind=\"invariant
    \"␣x=\"144\"␣y=\"168\">x&lt;=1␣&amp;&amp;\n"+
```

```
"cp[gtasknr-1]&lt;=dead-cr[gtasknr-1]+1</label></location><location id
    =\"id49\" x=\"160\" y=\"0\"><name x=\"150\" y=\"-30\">Running2</name
    ><label kind=\"invariant\" x=\"144\" y=\"8\">x&lt;1 &amp;&amp; \n"+
"cp[gtasknr-1]&lt;=dead-cr[gtasknr-1]+1</label></location><location id
    =\"id50\" x=\"-62\" y=\"1\"><name x=\"-72\" y=\"-29\">Running1</name
    ><urgent/></location><init ref=\"id46\"/><transition><source ref=\"
    id50\"/><target ref=\"id46\"/><label kind=\"synchronisation\" x
    =\"-240\" y=\"0\">tact[PRE]!</label><nail x=\"-256\" y=\"0\"/><nail
    x=\"-256\" y=\"-160\"/></transition><transition><source ref=\"id50
    \"/><target ref=\"id46\"/><label kind=\"guard\" x=\"-240\" y
    =\"-128\">cr[gtasknr-1]==0 &amp;&amp; \n"+
"cp[gtasknr-1]==ptime \n"+
"&amp;&amp; !pendMN(h_edf) \n"+
"&amp;&amp; !locked(l_out) \n"+
"&amp;&amp; !locked(h_t)</label><label kind=\"synchronisation\" x
    =\"-240\" y=\"-56\">tact[FINa]!</label><nail x=\"-200\" y
    =\"-80\"/></transition><transition><source ref=\"id46\"/><target ref
    =\"id50\"/><label kind=\"synchronisation\" x=\"-8\" y=\"-176\">tact[
    RUN2]?</label><nail x=\"96\" y=\"-160\"/><nail x=\"96\" y
    =\"-152\"/></transition><transition><source ref=\"id50\"/><target
    ref=\"id46\"/><label kind=\"guard\" x=\"-96\" y=\"-136\">cr[gtasknr
    -1]==0 &amp;&amp; \n"+
"cp[gtasknr-1]&lt;ptime &amp;&amp; \n"+
"!pendMN(h_edf) &amp;&amp; \n"+
"!locked(l_out) &amp;&amp; \n"+
"!locked(h_t)</label><label kind=\"synchronisation\" x=\"-96\" y
    =\"-64\">tact[FIN]!</label></transition><transition><source ref=\"
    id47\"/><target ref=\"id50\"/><label kind=\"guard\" x=\"-8\" y
    =\"56\">cr[gtasknr-1]&gt;0</label><label kind=\"assignment\" x
    =\"-24\" y=\"72\">l_in[schnr-1]=false, \n"+
"h_t[schnr-1]=false</label><nail x=\"0\" y=\"80\"/></transition><
    transition><source ref=\"id47\"/><target ref=\"id50\"/><label kind
    =\"guard\" x=\"-160\" y=\"56\">cr[gtasknr-1]==0</label><label kind
    =\"assignment\" x=\"-200\" y=\"72\">pending[gtasknr-1]=true, \n"+
"l_in[schnr-1]=false, \n"+
"h_fin[schnr-1]=true, \n"+
"h_t[schnr-1]=false</label><nail x=\"-128\" y=\"80\"/></transition><
    transition><source ref=\"id48\"/><target ref=\"id47\"/><label kind
    =\"guard\" x=\"-40\" y=\"144\">x==1</label><label kind=\"assignment
    \" x=\"-40\" y=\"160\">cr[gtasknr-1]--, edf[tasknr-1]--, \n"+
"nowrun=false</label></transition><transition><source ref=\"id49\"/><
    target ref=\"id48\"/><label kind=\"guard\" x=\"160\" y=\"56\">x&gt
    ;0</label><label kind=\"assignment\" x=\"160\" y=\"72\">h_t[schnr
    -1]=true</label></transition><transition><source ref=\"id50\"/><
    target ref=\"id49\"/><label kind=\"guard\" x=\"16\" y=\"-80\">cr[
    gtasknr-1]&gt;0 &amp;&amp; \n"+
"!pendMN(pending)\n"+
"&amp;&amp; !locked(l_out)\n"+
"&amp;&amp; !locked(h_t)\n"+
"&amp;&amp; !locked(h_r)</label><label kind=\"assignment\" x=\"16\" y
    =\"0\">x=0, \n"+
"l_in[schnr-1]=true, \n"+
"nowrun=true"+
"</label></transition></template><system>\n"+
toStringAll(application.tasks, platform.processors)+
"\n\n"+
"system "+toStringSystem(application.tasks, platform.processors)+
        ";</system></nta>";
    }
}
```

Appendix C

Source code for single processor examples

Single processor example 1 using RM

```
import java.io.*;

public class single_ex1_rm{
    public static void main(String[] args){
        String fl = "single_ex1_rm";
        Task t1 = new Task(6, 10, 0, 10, 1, 1);
        Task t2 = new Task(6, 20, 0, 20, 2, 1);
        Task t3 = new Task(2, 30, 0, 30, 3, 1);

        Processor p1 = new Processor(1, Processor.RM);

        Task[][] tasks = { { t1,t2,t3} };

        Processor[] ps = {p1};
        Platform platform = new Platform(ps);
        Application apps = new Application(tasks);

        MPSoC sys = new MPSoC(apps,platform);
        try{
            PrintStream fout = new PrintStream(new File(fl+".xml"));
            fout.println(sys.mkXML());
        }
        catch(Exception e){e.printStackTrace();}
    }
}
```

Single processor example 1 using EDF

```
import java.io.*;

public class single_ex1_edf{
    public static void main(String[] args){
        String fl = "single_ex1_edf";
        Task t1 = new Task(6, 10, 0, 10, 1, 1);
        Task t2 = new Task(6, 20, 0, 20, 2, 1);
        Task t3 = new Task(2, 30, 0, 30, 3, 1);

        Processor p1 = new Processor(1, Processor.EDF);

        Task[][] tasks = { { t1,t2,t3} };

        Processor[] ps = {p1};
        Platform platform = new Platform(ps);
        Application apps = new Application(tasks);

        MPSoC sys = new MPSoC(apps,platform);
        try{
            PrintStream fout = new PrintStream(new File(fl+".xml"));
            fout.println(sys.mkXML());
        }
        catch(Exception e){e.printStackTrace();}
    }
}
```

Single processor example 2 using RM

```
import java.io.*;

public class single_ex2_rm{
    public static void main(String[] args){
        String fl = "single_ex2_rm";
        Task t1 = new Task(6, 10, 0, 10, 1, 1);
        Task t2 = new Task(6, 20, 0, 20, 2, 1);
        Task t3 = new Task(3, 30, 0, 30, 3, 1);

        Processor p1 = new Processor(1, Processor.RM);

        Task[][] tasks = { { t1,t2,t3} };

        Processor[] ps = {p1};
        Platform platform = new Platform(ps);
        Application apps = new Application(tasks);

        MPSoC sys = new MPSoC(apps,platform);
        try{
            PrintStream fout = new PrintStream(new File(fl+".xml"));
            fout.println(sys.mkXML());
        }
        catch(Exception e){e.printStackTrace();}
    }
}
```

Single processor example 2 using EDF

```
import java.io.*;

public class single_ex2_edf{
    public static void main(String[] args){
        String fl = "single_ex2_edf";
        Task t1 = new Task(6, 10, 0, 10, 1, 1);
        Task t2 = new Task(6, 20, 0, 20, 2, 1);
        Task t3 = new Task(3, 30, 0, 30, 3, 1);

        Processor p1 = new Processor(1, Processor.EDF);

        Task[][] tasks = { { t1,t2,t3} };

        Processor[] ps = {p1};
        Platform platform = new Platform(ps);
        Application apps = new Application(tasks);

        MPSoC sys = new MPSoC(apps,platform);
        try{
            PrintStream fout = new PrintStream(new File(fl+".xml"));
            fout.println(sys.mkXML());
        }
        catch(Exception e){e.printStackTrace();}
    }
}
```

Single processor example 3 using RM

```
import java.io.*;

public class single_ex3_rm{
    public static void main(String[] args){
        String fl = "single_ex3_rm";
        Task t1 = new Task(6, 10, 0, 10, 1, 1);
        Task t2 = new Task(6, 20, 0, 20, 2, 1);
        Task t3 = new Task(4, 30, 0, 30, 3, 1);

        Processor p1 = new Processor(1, Processor.RM);

        Task[][] tasks = { { t1,t2,t3} };

        Processor[] ps = {p1};
        Platform platform = new Platform(ps);
        Application apps = new Application(tasks);

        MPSoC sys = new MPSoC(apps,platform);
        try{
            PrintStream fout = new PrintStream(new File(fl+".xml"));
            fout.println(sys.mkXML());
        }
        catch(Exception e){e.printStackTrace();}
    }
}
```

Single processor example 3 using EDF

```
import java.io.*;
```

```
public class single_ex3_edf{
    public static void main(String[] args){
        String fl = "single_ex3_edf";
        Task t1 = new Task(6, 10, 0, 10, 1, 1);
        Task t2 = new Task(6, 20, 0, 20, 2, 1);
        Task t3 = new Task(4, 30, 0, 30, 3, 1);

        Processor p1 = new Processor(1, Processor.EDF);

        Task[][] tasks = { { t1,t2,t3} };

        Processor[] ps = {p1};
        Platform platform = new Platform(ps);
        Application apps = new Application(tasks);

        MPSoC sys = new MPSoC(apps,platform);
        try{
            PrintStream fout = new PrintStream(new File(fl+".xml"));
            fout.println(sys.mkXML());
        }
        catch(Exception e){e.printStackTrace();}
    }
}
```

Single processor example 4 using RM

```
import java.io.*;

public class single_ex4_rm{
    public static void main(String[] args){
        String fl = "single_ex4_rm";
        Task t1 = new Task(3, 5, 0, 5, 1, 1);
        Task t2 = new Task(2, 4, 0, 6, 2, 1);

        Processor p1 = new Processor(1, Processor.RM);

        Task[][] tasks = { { t1,t2} };

        Processor[] ps = {p1};
        Platform platform = new Platform(ps);
        Application apps = new Application(tasks);

        MPSoC sys = new MPSoC(apps,platform);
        try{
            PrintStream fout = new PrintStream(new File(fl+".xml"));
            fout.println(sys.mkXML());
        }
        catch(Exception e){e.printStackTrace();}
    }
}
```

Single processor example 4 using DM

```
import java.io.*;

public class single_ex4_dm{
    public static void main(String[] args){
        String fl = "single_ex4_dm";
```

```
        Task t1 = new Task(3, 5, 0, 5, 1, 1);
        Task t2 = new Task(2, 4, 0, 6, 2, 1);

        Processor p1 = new Processor(1, Processor.DM);

        Task[][] tasks = { { t1,t2} };

        Processor[] ps = {p1};
        Platform platform = new Platform(ps);
        Application apps = new Application(tasks);

        MPSoC sys = new MPSoC(apps,platform);
        try{
            PrintStream fout = new PrintStream(new File(fl+".xml"));
            fout.println(sys.mkXML());
        }
        catch(Exception e){e.printStackTrace();}
    }
}
```

Single processor example 5 using RM

```
import java.io.*;

public class single_ex5_rm{
    public static void main(String[] args){
        String fl = "single_ex5_rm";
        Task t1 = new Task(3, 15, 0, 15, 1, 1);
        Task t2 = new Task(2, 10, 0, 10, 2, 1);
        Task t3 = new Task(4, 35, 0, 35, 3, 1);
        Task t4 = new Task(5, 13, 0, 13, 4, 1);

        Processor p1 = new Processor(1, Processor.RM);

        Task[][] tasks = { { t1,t2,t3,t4} };

        Processor[] ps = {p1};
        Platform platform = new Platform(ps);
        Application apps = new Application(tasks);

    apps.addDep(1,3);

        MPSoC sys = new MPSoC(apps,platform);
        try{
            PrintStream fout = new PrintStream(new File(fl+".xml"));
            fout.println(sys.mkXML());
        }
        catch(Exception e){e.printStackTrace();}
    }
}
```

Single processor example 5 using EDF

```
import java.io.*;

public class single_ex5_edf{
    public static void main(String[] args){
        String fl = "single_ex5_edf";
```

```
        Task t1 = new Task(3, 15, 0, 15, 1, 1);
        Task t2 = new Task(2, 10, 0, 10, 2, 1);
        Task t3 = new Task(4, 35, 0, 35, 3, 1);
        Task t4 = new Task(5, 13, 0, 13, 4, 1);

        Processor p1 = new Processor(1, Processor.EDF);

        Task[][] tasks = { { t1,t2,t3,t4} };

        Processor[] ps = {p1};
        Platform platform = new Platform(ps);
        Application apps = new Application(tasks);

    apps.addDep(1,3);

        MPSoC sys = new MPSoC(apps,platform);
        try{
            PrintStream fout = new PrintStream(new File(fl+".xml"));
            fout.println(sys.mkXML());
        }
        catch(Exception e){e.printStackTrace();}
    }
}
```

Appendix D

Source code for multiprocessor examples

Multiprocessor example 1 using RM

```
import java.io.*;

public class multi_ex1_rm{
    public static void main(String[] args){
        String fl = "multi_ex1_rm";
        Task t1 = new Task(2, 4, 0, 4, 1, 1);
        Task t2 = new Task(2, 6, 0, 6, 2, 1);
        Task t3 = new Task(2, 6, 0, 6, 3, 2);
        Task t4 = new Task(3, 6, 4, 6, 4, 2);

        Processor p1 = new Processor(1, Processor.RM);
        Processor p2 = new Processor(2, Processor.RM);

        Task[][] tasks = { { t1, t2} , { t3, t4 } };

        Processor[] ps = {p1,p2};
        Platform platform = new Platform(ps);
        Application apps = new Application(tasks);

        apps.addDep(2,3);

        MPSoC sys = new MPSoC(apps,platform);
        try{
            PrintStream fout = new PrintStream(new File(fl+".xml"));
            fout.println(sys.mkXML());
        }
        catch(Exception e){e.printStackTrace();}
```

```
    }
}
```

Multiprocessor example 1 using EDF

```
import java.io.*;

public class multi_ex1_edf{
    public static void main(String[] args){
        String fl = "multi_ex1_edf";
        Task t1 = new Task(2, 4, 0, 4, 1, 1);
        Task t2 = new Task(2, 6, 0, 6, 2, 1);
        Task t3 = new Task(2, 6, 0, 6, 3, 2);
        Task t4 = new Task(3, 6, 4, 6, 4, 2);

        Processor p1 = new Processor(1, Processor.EDF);
        Processor p2 = new Processor(2, Processor.EDF);

        Task[][] tasks = { { t1, t2} , { t3, t4 } };

        Processor[] ps = {p1,p2};
        Platform platform = new Platform(ps);
        Application apps = new Application(tasks);

        apps.addDep(2,3);

        MPSoC sys = new MPSoC(apps,platform);
        try{
            PrintStream fout = new PrintStream(new File(fl+".xml"));
            fout.println(sys.mkXML());
        }
        catch(Exception e){e.printStackTrace();}
    }
}
```

Multiprocessor example 2 using RM

```
import java.io.*;

public class multi_ex2_rm{
    public static void main(String[] args){
        String fl = "multi_ex2_rm";
        Task t1 = new Task(2, 4, 0, 4, 1, 1);
        Task t2 = new Task(1, 6, 0, 6, 2, 1);
        Task t3 = new Task(1, 6, 0, 6, 3, 2);
        Task t4 = new Task(2, 6, 0, 6, 4, 3);
        Task t5 = new Task(3, 6, 4, 6, 5, 3);

        Processor p1 = new Processor(1, Processor.RM);
        Processor p2 = new Processor(2, Processor.RM);
        Processor p3 = new Processor(3, Processor.RM);

        Task[][] tasks = { { t1, t2} , {t3}, { t4, t5 } };

        Processor[] ps = {p1,p2,p3};
        Platform platform = new Platform(ps);
        Application apps = new Application(tasks);
```

```
        apps.addDep(2,3);
        apps.addDep(3,4);

        MPSoC sys = new MPSoC(apps,platform);
        try{
            PrintStream fout = new PrintStream(new File(fl+".xml"));
            fout.println(sys.mkXML());
        }
        catch(Exception e){e.printStackTrace();}
    }
}
```

Multiprocessor example 2 using EDF

```
import java.io.*;

public class multi_ex2_edf{
    public static void main(String[] args){
        String fl = "multi_ex2_edf";
        Task t1 = new Task(2, 4, 0, 4, 1, 1);
        Task t2 = new Task(1, 6, 0, 6, 2, 1);
        Task t3 = new Task(1, 6, 0, 6, 3, 2);
        Task t4 = new Task(2, 6, 0, 6, 4, 3);
        Task t5 = new Task(3, 6, 4, 6, 5, 3);

        Processor p1 = new Processor(1, Processor.EDF);
        Processor p2 = new Processor(2, Processor.EDF);
        Processor p3 = new Processor(3, Processor.EDF);

        Task[][] tasks = { { t1, t2} , {t3}, { t4, t5 } };

        Processor[] ps = {p1,p2,p3};
        Platform platform = new Platform(ps);
        Application apps = new Application(tasks);

        apps.addDep(2,3);
        apps.addDep(3,4);

        MPSoC sys = new MPSoC(apps,platform);
        try{
            PrintStream fout = new PrintStream(new File(fl+".xml"));
            fout.println(sys.mkXML());
        }
        catch(Exception e){e.printStackTrace();}
    }
}
```

Multiprocessor example 3 using RM

```
import java.io.*;

public class multi_ex3_rm{
    public static void main(String[] args){
        String fl = "multi_ex3_rm";
        Task t1 = new Task(2, 40, 0, 40, 1, 1);
        Task t2 = new Task(2, 40, 0, 40, 2, 1);
        Task t3 = new Task(2, 40, 0, 40, 3, 2);
        Task t4 = new Task(2, 40, 0, 40, 4, 2);
```

```
        Task t5 = new Task(2, 40, 0, 40, 5, 3);
        Task t6 = new Task(2, 40, 0, 40, 6, 3);
        Task t7 = new Task(2, 40, 0, 40, 7, 4);
        Task t8 = new Task(2, 40, 0, 40, 8, 4);
        Task t9 = new Task(2, 40, 0, 40, 9, 5);
        Task t10 = new Task(2, 40, 0, 40, 10, 6);

        Processor p1 = new Processor(1, Processor.RM);
        Processor p2 = new Processor(2, Processor.RM);
        Processor p3 = new Processor(3, Processor.RM);
        Processor p4 = new Processor(4, Processor.RM);
        Processor p5 = new Processor(5, Processor.RM);
        Processor p6 = new Processor(6, Processor.RM);

        Task[][] tasks = {{t1,t2},{t3,t4},{t5,t6},{t7,t8},{t9},{t10}};

        Processor[] ps = {p1,p2,p3,p4,p5,p6};
        Platform platform = new Platform(ps);
        Application apps = new Application(tasks);

        apps.addDep(2,3);

        MPSoC sys = new MPSoC(apps,platform);
        try{
            PrintStream fout = new PrintStream(new File(fl+".xml"));
            fout.println(sys.mkXML());
        }
        catch(Exception e){e.printStackTrace();}
    }
}
```

Multiprocessor example 3 using DM

```
import java.io.*;

public class multi_ex3_dm{
    public static void main(String[] args){
        String fl = "multi_ex3_dm";
        Task t1 = new Task(2, 40, 0, 40, 1, 1);
        Task t2 = new Task(2, 40, 0, 40, 2, 1);
        Task t3 = new Task(2, 40, 0, 40, 3, 2);
        Task t4 = new Task(2, 40, 0, 40, 4, 2);
        Task t5 = new Task(2, 40, 0, 40, 5, 3);
        Task t6 = new Task(2, 40, 0, 40, 6, 3);
        Task t7 = new Task(2, 40, 0, 40, 7, 4);
        Task t8 = new Task(2, 40, 0, 40, 8, 4);
        Task t9 = new Task(2, 40, 0, 40, 9, 5);
        Task t10 = new Task(2, 40, 0, 40, 10, 6);

        Processor p1 = new Processor(1, Processor.DM);
        Processor p2 = new Processor(2, Processor.DM);
        Processor p3 = new Processor(3, Processor.DM);
        Processor p4 = new Processor(4, Processor.DM);
        Processor p5 = new Processor(5, Processor.DM);
        Processor p6 = new Processor(6, Processor.DM);

        Task[][] tasks = {{t1,t2},{t3,t4},{t5,t6},{t7,t8},{t9},{t10}};

        Processor[] ps = {p1,p2,p3,p4,p5,p6};
        Platform platform = new Platform(ps);
        Application apps = new Application(tasks);
```

```
        apps.addDep(2,3);

        MPSoC sys = new MPSoC(apps,platform);
        try{
            PrintStream fout = new PrintStream(new File(fl+".xml"));
            fout.println(sys.mkXML());
        }
        catch(Exception e){e.printStackTrace();}
    }
}
```

Multiprocessor example 3 using EDF

```
import java.io.*;

public class multi_ex3_edf{
    public static void main(String[] args){
        String fl = "multi_ex3_edf";
        Task t1 = new Task(2, 40, 0, 40, 1, 1);
        Task t2 = new Task(2, 40, 0, 40, 2, 1);
        Task t3 = new Task(2, 40, 0, 40, 3, 2);
        Task t4 = new Task(2, 40, 0, 40, 4, 2);
        Task t5 = new Task(2, 40, 0, 40, 5, 3);
        Task t6 = new Task(2, 40, 0, 40, 6, 3);
        Task t7 = new Task(2, 40, 0, 40, 7, 4);
        Task t8 = new Task(2, 40, 0, 40, 8, 4);
        Task t9 = new Task(2, 40, 0, 40, 9, 5);
        Task t10 = new Task(2, 40, 0, 40, 10, 6);

        Processor p1 = new Processor(1, Processor.EDF);
        Processor p2 = new Processor(2, Processor.EDF);
        Processor p3 = new Processor(3, Processor.EDF);
        Processor p4 = new Processor(4, Processor.EDF);
        Processor p5 = new Processor(5, Processor.EDF);
        Processor p6 = new Processor(6, Processor.EDF);

        Task[][] tasks = {{t1,t2},{t3,t4},{t5,t6},{t7,t8},{t9},{t10}};

        Processor[] ps = {p1,p2,p3,p4,p5,p6};
        Platform platform = new Platform(ps);
        Application apps = new Application(tasks);

        apps.addDep(2,3);

        MPSoC sys = new MPSoC(apps,platform);
        try{
            PrintStream fout = new PrintStream(new File(fl+".xml"));
            fout.println(sys.mkXML());
        }
        catch(Exception e){e.printStackTrace();}
    }
}
```

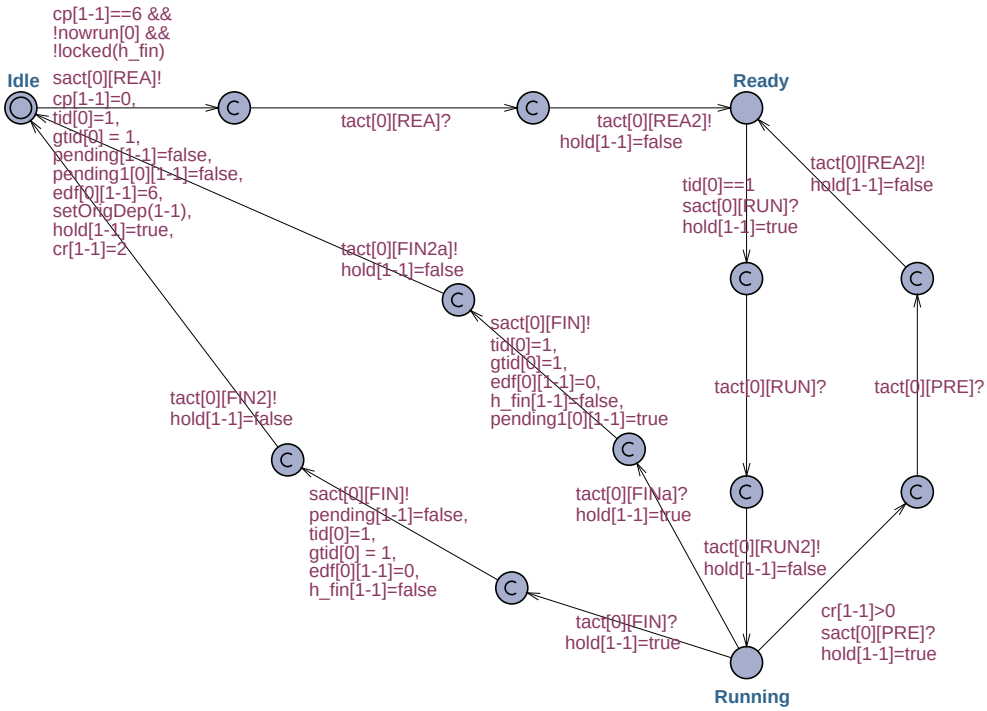
Appendix E

Full timed-automata model for example

In the following, the full timed-automata model for the example specified in section 5.3.6 is given. This is done in terms of each timed automaton and their local declarations, the global declarations and the system declaration.

Task1 automata

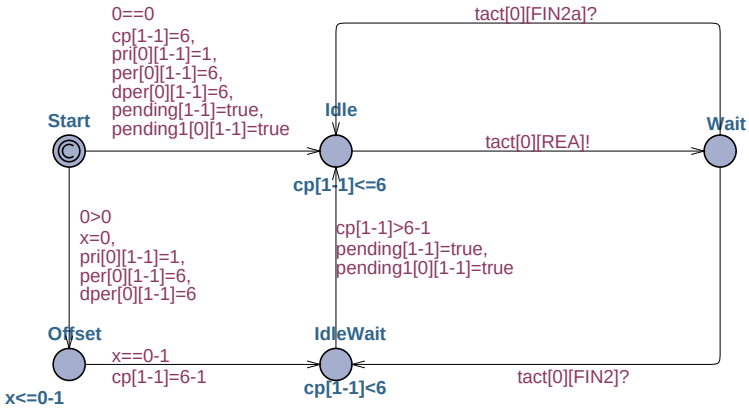
TaskControl1



Local declarations for TaskControl1

none

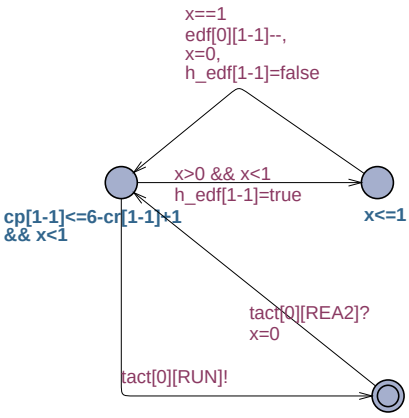
Idle1



Local declarations for Idle1

```
clock x;
```

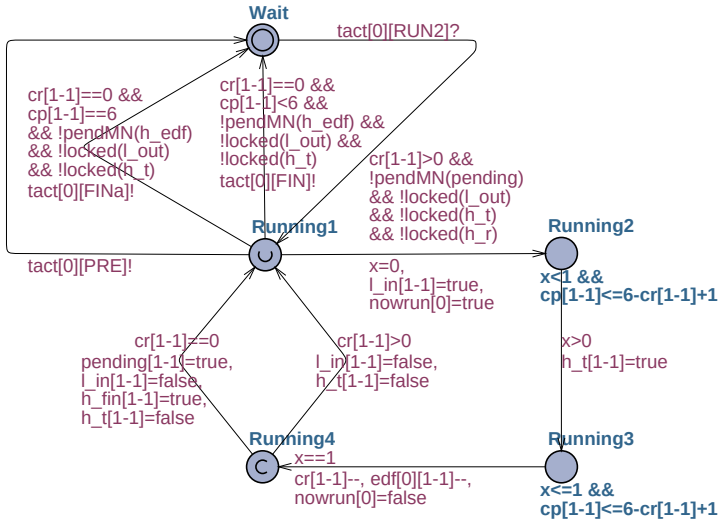
Ready1



Local declarations for Ready1

```
clock x;
```

Running1

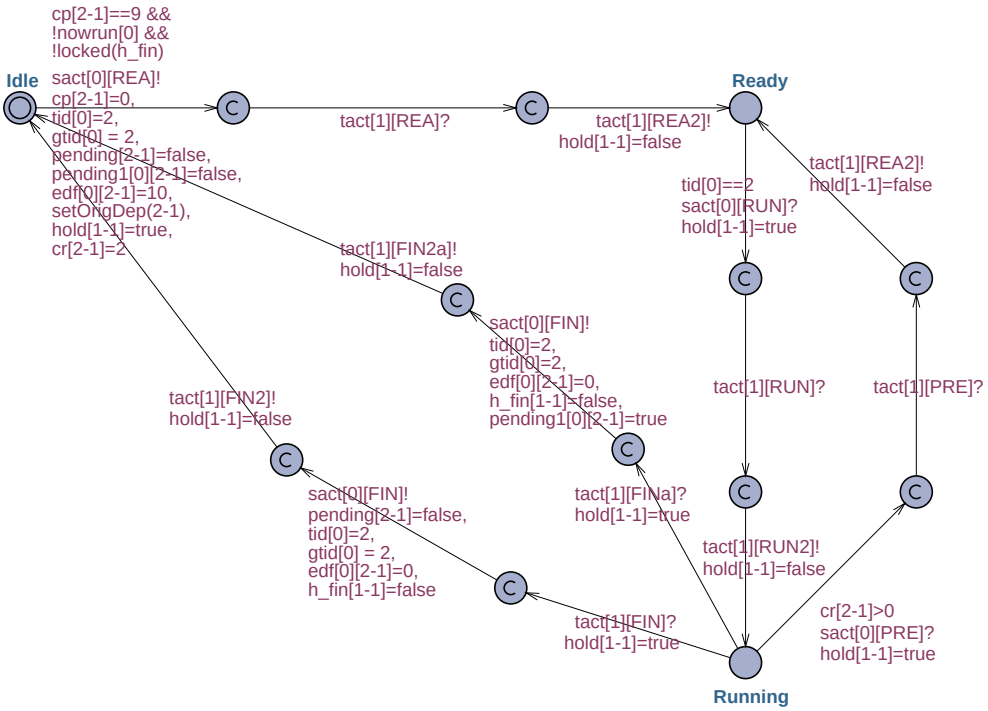


Local declarations for Running1

```
clock x;
```

Task2 automata

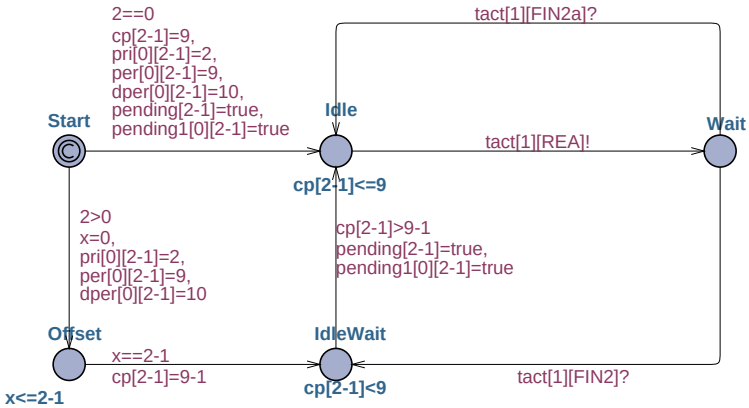
TaskControl2



Local declarations for TaskControl2

none

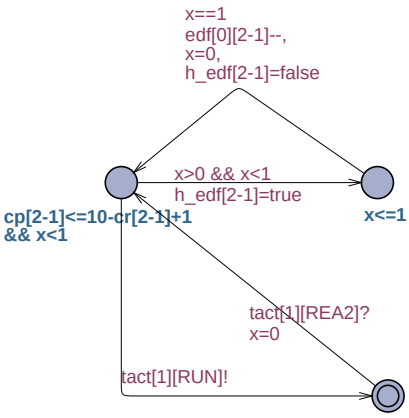
Idle2



Local declarations for Idle2

```
clock x;
```

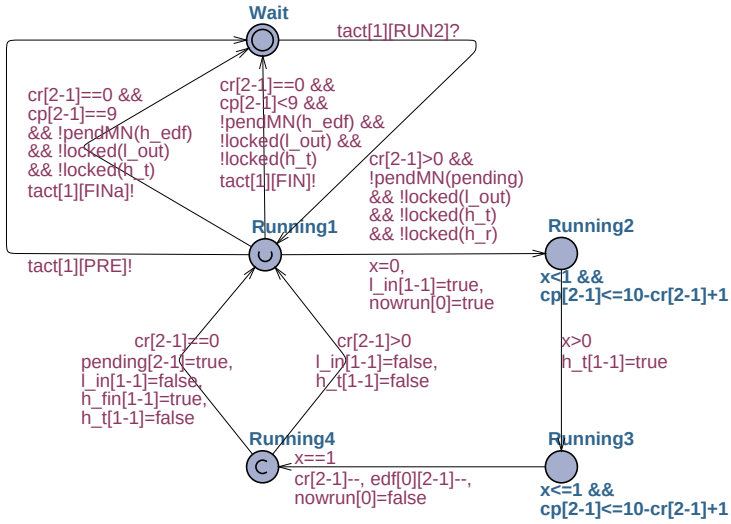
Ready2



Local declarations for Ready2

```
clock x;
```

Running2

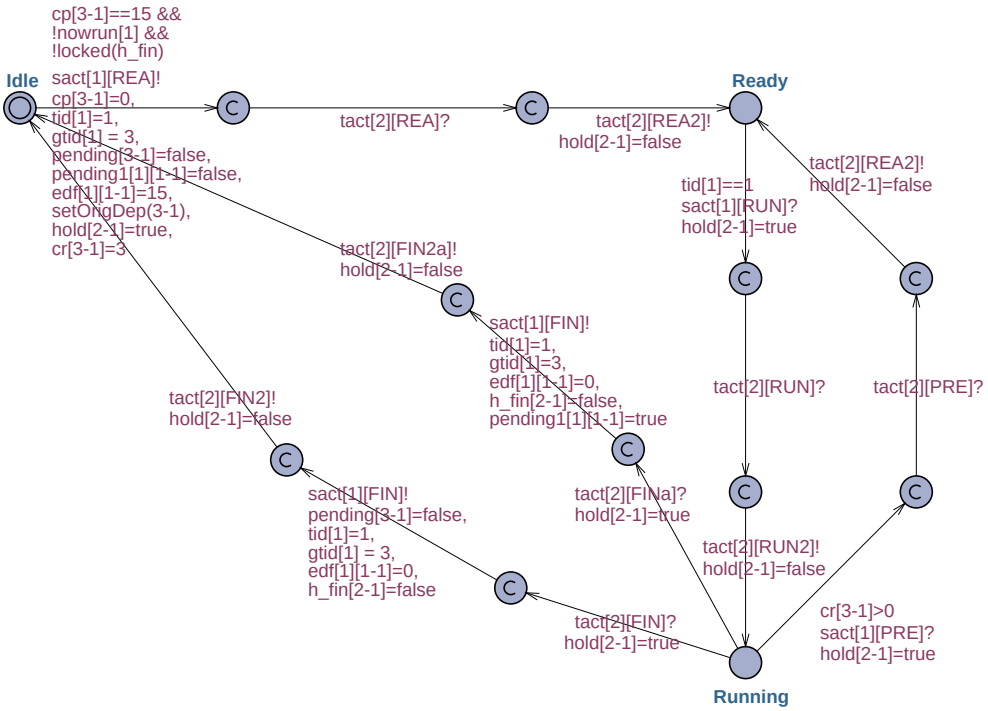


Local declarations for Running2

```
clock x;
```

Task3 automata

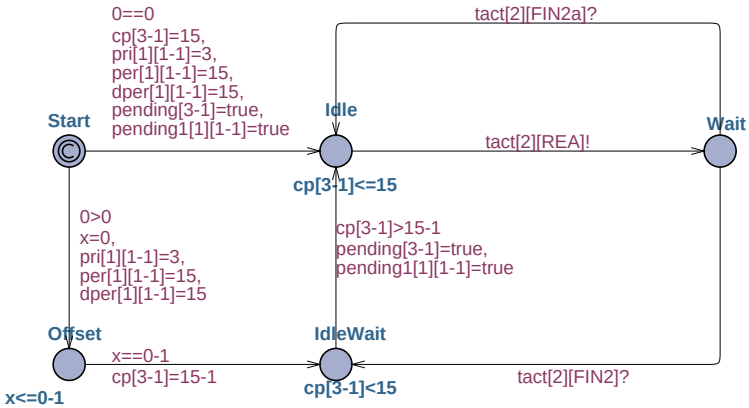
TaskControl3



Local declarations for TaskControl3

none

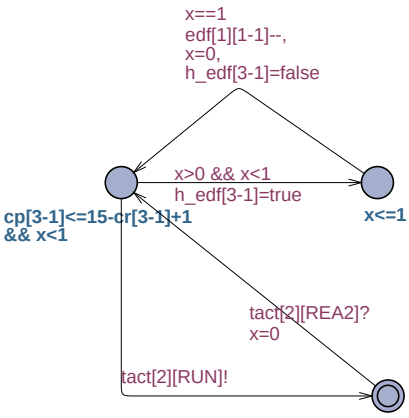
Idle3



Local declarations for Idle3

```
clock x;
```

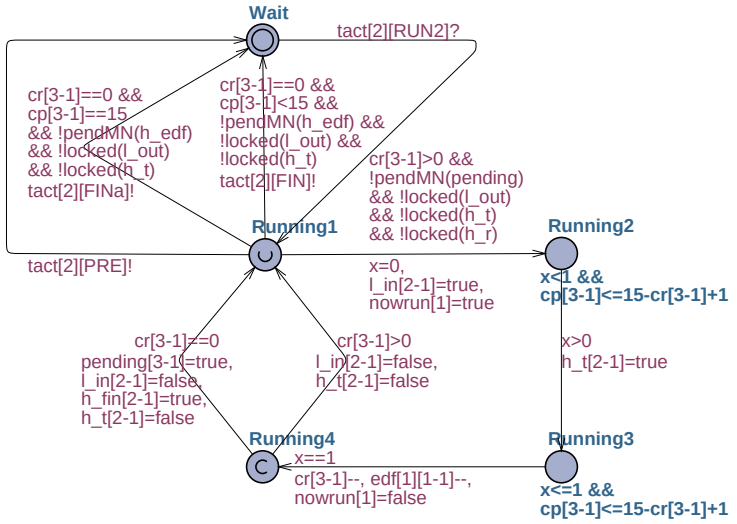
Ready3



Local declarations for Ready3

```
clock x;
```

Running3

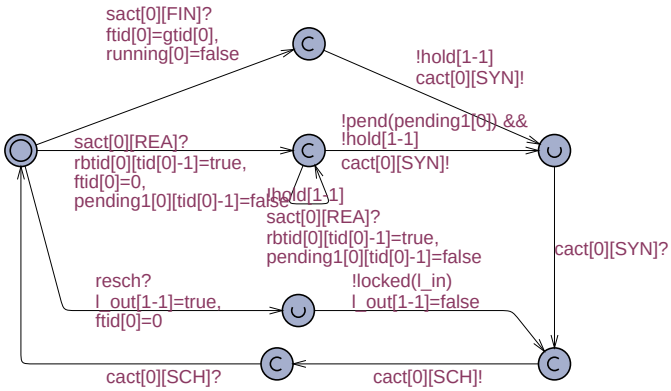


Local declarations for Running3

```
clock x;
```

Processor1 automata

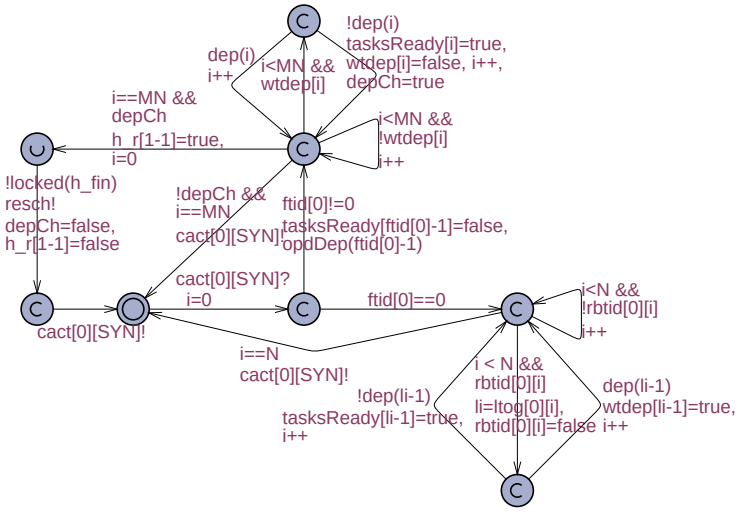
Controller1



Local declarations for Controller1

none

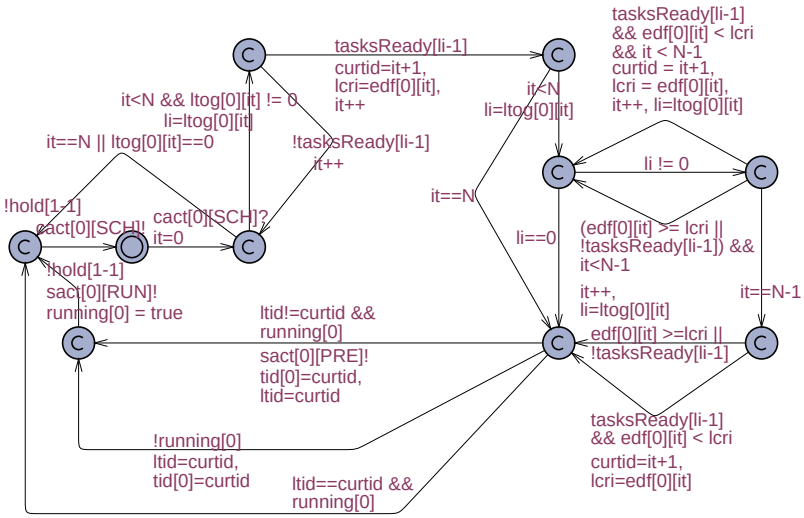
Synchronizer1



Local declarations for Synchronizer1

```
int[0,MN] i; //iteration variable
int[0,MN] li; //variable used to hold global id for tasks
bool depCh; //flag used if a dependency has been changed
```

Scheduler1

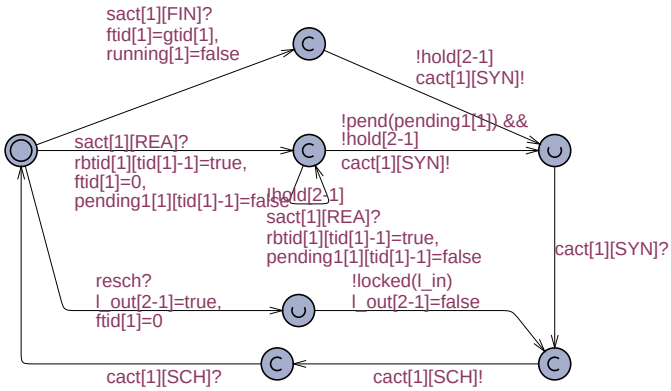


Local declarations for Scheduler1

```
int[0,N] it; //iteration variable
int lcri; //variable used to hold the criterion of the task currently chosen
int[0,N] ltid; //variable used to hold the id of the task currently running
int[0,N] curtid; //variable used to hold the id of the task currently chosen
int[0,MN] li; //variable used to hold global id for tasks
```

Processor2 automata

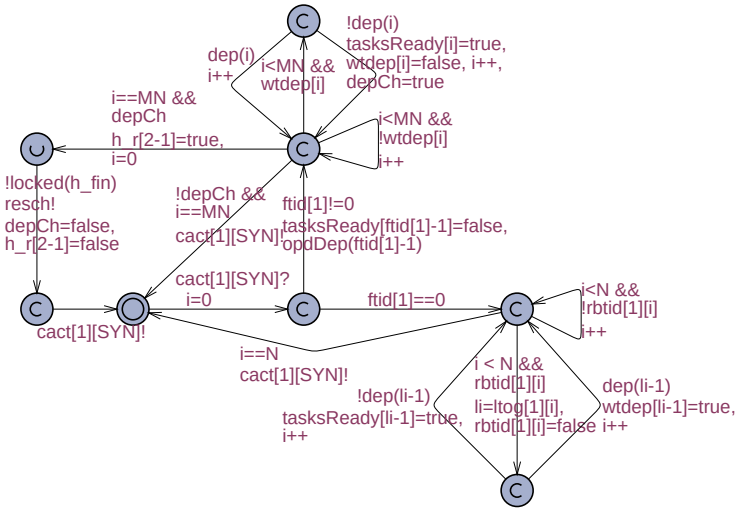
Controller2



Local declarations for Controller2

none

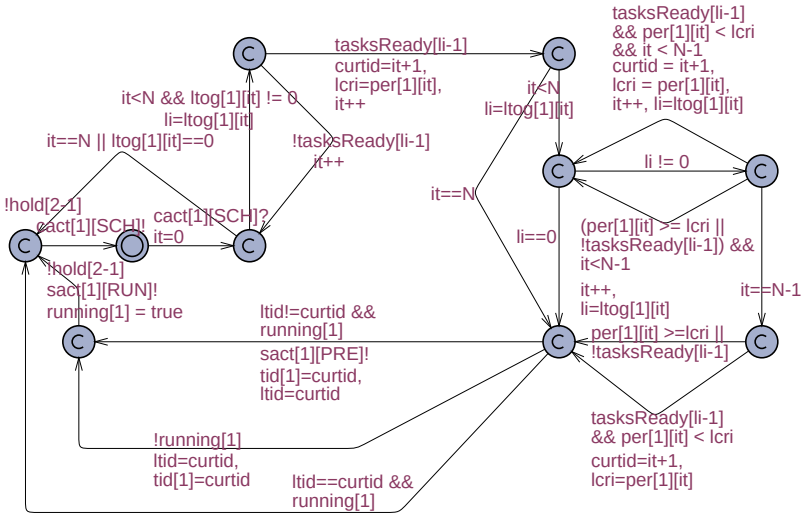
Synchronizer2



Local declarations for Synchronizer2

```
int[0,MN] i; //iteration variable
int[0,MN] li; //variable used to hold global id for tasks
bool depCh; //flag used if a dependency has been changed
```

Scheduler2



Local declarations for Scheduler2

```
int[0,N] it; //iteration variable
int lcri; //variable used to hold the criterion of the task currently chosen
int[0,N] ltid; //variable used to hold the id of the task currently running
int[0,N] curtid; //variable used to hold the id of the task currently chosen
int[0,MN] li; //variable used to hold global id for tasks
```

Global declarations

```
const int M = 2; //The number of processors
const int N = 2; //The maximum number of tasks per Processor
const int MN = 3; //The total number of tasks
//symbolic representation of application-to-platform- and internal task actions
const int REA = 0, RUN = 1, PRE = 2, FIN = 3, FIN2 = 4, RUN2 = 5, REA2 = 6, FINa = 7, FIN2a = 8;
const int SYN = 0, SCH = 1; //symbolic representation of internal processor synchronizing
int[0,MN] ltog[M][N]={{1, 2}, {3, 0}}; //global taskids from locals
broadcast chan resch; //broadcast channel for rescheluling after a task has finished
chan cact[M][2]; //channel array for internal processor actions
chan sact[M][4]; //channel array for scheduling actions
chan tact[MN][9]; //channel array for internal task synchronization

clock cp[MN]; //array of period clocks for the tasks
int cr[MN]; // array of variables containing the running time for the tasks

int[0,N] tid[M]; //transfer of local taskid from task to controller
int[0,MN] gtid[M]; //transfer of global taskid from task to controller
int[0,MN] ftid[M]; //taskid of task which has finished, ftid=0 means no finished task
bool rbtid[M][N]; //array of tasks which have become ready
bool running[M]; //indicating wether a task is currently running on the processor
bool tasksReady[MN]; //array of tasks which are ready for scheduling


//Criteria usable in scheduling
int pri[M][N]; //criterion used for user defined priority scheduling
int per[M][N]; //criterion used for rate monotonic scheduling
int dper[M][N]; //criterion used for deadline monotonic scheduling
int edf[M][N]; //criterion used for earliest deadline first scheduling

//array for original dependencies, 1 for dependency, 0 for no dependency
bool origdep[MN][MN]={{0,0,0},{0,0,0},{0,1,0}};

//dynamically updated array for current dependencies
bool depend[MN][MN]={{0,0,0},{0,0,0},{0,1,0}};

bool wtdep[MN]; //array for tasks ready but waiting for dependencies to be resolved

//Locking mechanisms
bool nowrun[M]; //ensuring completion of 'runs' before reacting on ready
bool pending[MN]; //ensuring global wait for finish & ready before next 'run'
bool pending1[M][N]; //ensuring reaction on all ready signals available locally
bool h_edf[MN]; //ensuring synchronization on extra state in ready for edf
bool h_t[M]; //ensuring completion of all 'runs' before next 'run'
bool h_fin[M]; //ensuring reaction on finish before ready
bool l_in[M]; //ensuring completion of local scheduling before glocal reschedule
bool l_out[M]; //ensuring reaction on reschedule before next 'run'
bool hold[M]; //ensuring 3-in-one completion of transitions on Task_Control
bool h_r[M]; //ensuring reschedule before next 'run'

//function checking for dependencies for task t
bool dep(int t)
{
  for (ini : int[0,MN-1])
  {    if (depend[t][ini])
    {
      return true;
    }
  }
  return false;
}
//function updating dependencies when task t has finished
```

```
void opdDep(int t)
{
  for (ini : int[0,MN-1])
  {
    depend[ini][t]=false;
  }
}
//function setting the original dependency values for task t
void setOrigDep(int t)
{
  for (ini : int[0,MN-1])
  {
    depend[t][ini]=origdep[t][ini];
  }
}
//function checking for existance of boolean value in array of size M
bool locked(bool la[M])
{
  for (ini : int[0,M-1])
  {    if (la[ini])
    {
      return true;
    }
  }
  return false;
}
//function checking for existance of boolean value in array of size N
bool pend(bool pen[N])
{
  bool b = false;
  for (ini : int[0,N-1])
  {
    if (pen[ini] == true)
      {
        return true;
      }
  }
  return false;
}
//function checking for existance of boolean value in array of size MN
bool pendMN(bool pen[MN])
{
  bool b = false;
  for (ini : int[0,MN-1])
  {
    if (pen[ini] == true)
      {
        return true;
      }
  }
  return false;
}
```

System declarations

```
//Processor1
Con1 = Control(1, sact[0], ftid[0], tid[0], gtid[0], rbtid[0],
running[0], cact[0], pending1[0]);

Syn1 = Synchron(1, ftid[0], rbtid[0], cact[0], ltog[0]);

Sch1 = SchedMin(1, tid[0], edf[0], running[0], ftid[0], sact[0],
cact[0], ltog[0]);


//Task1
Taskc1 = Task_Control(1, 1, 1, 2, 6, 0, 6, 1, tid[0], gtid[0], pri[0],
per[0], dper[0], nowrun[0], sact[0], edf[0], pending1[0],tact[0]);

Taski1 = Task_Idle(1, 1, 1, 2, 6, 0, 6, 1, tid[0], gtid[0], pri[0],
per[0], dper[0], nowrun[0], sact[0], edf[0], pending1[0],tact[0]);

Taskre1 = Task_Ready(1, 1, 1, 2, 6, 0, 6, 1, tid[0], gtid[0], pri[0],
per[0], dper[0], nowrun[0], sact[0], edf[0], pending1[0],tact[0]);

Taskru1 = Task_Running(1, 1, 1, 2, 6, 0, 6, 1, tid[0], gtid[0],
pri[0], per[0], dper[0], nowrun[0], sact[0], edf[0],
pending1[0],tact[0]);


//Task2
Taskc2 = Task_Control(1, 2, 2, 2, 10, 2, 9, 2, tid[0], gtid[0],
pri[0], per[0], dper[0], nowrun[0], sact[0], edf[0],
pending1[0],tact[1]);

Taski2 = Task_Idle(1, 2, 2, 2, 10, 2, 9, 2, tid[0], gtid[0], pri[0],
per[0], dper[0], nowrun[0], sact[0], edf[0], pending1[0],tact[1]);

Taskre2 = Task_Ready(1, 2, 2, 2, 10, 2, 9, 2, tid[0], gtid[0], pri[0],
per[0], dper[0], nowrun[0], sact[0], edf[0], pending1[0],tact[1]);

Taskru2 = Task_Running(1, 2, 2, 2, 10, 2, 9, 2, tid[0], gtid[0],
pri[0], per[0], dper[0], nowrun[0], sact[0], edf[0],
pending1[0],tact[1]);


//Processor2
Con2 = Control(2, sact[1], ftid[1], tid[1], gtid[1], rbtid[1],
running[1], cact[1], pending1[1]);

Syn2 = Synchron(2, ftid[1], rbtid[1], cact[1], ltog[1]);

Sch2 = SchedMin(2, tid[1], per[1], running[1], ftid[1], sact[1],
cact[1], ltog[1]);


//Task3
Taskc3 = Task_Control(2, 3, 1, 3, 15, 0, 15, 3, tid[1], gtid[1],
pri[1], per[1], dper[1], nowrun[1], sact[1], edf[1],
pending1[1],tact[2]);

Taski3 = Task_Idle(2, 3, 1, 3, 15, 0, 15, 3, tid[1], gtid[1], pri[1],
per[1], dper[1], nowrun[1], sact[1], edf[1], pending1[1],tact[2]);

Taskre3 = Task_Ready(2, 3, 1, 3, 15, 0, 15, 3, tid[1], gtid[1],
pri[1], per[1], dper[1], nowrun[1], sact[1], edf[1],
```

```
pending1[1],tact[2]);

Taskru3 = Task_Running(2, 3, 1, 3, 15, 0, 15, 3, tid[1], gtid[1],
pri[1], per[1], dper[1], nowrun[1], sact[1], edf[1],
pending1[1],tact[2]);


//System
system Taskc1, Taski1, Taskre1, Taskru1,Taskc2, Taski2, Taskre2,
Taskru2,Taskc3, Taski3, Taskre3, Taskru3,Con1, Syn1, Sch1, Con2, Syn2,
Sch2;
```

Appendix F

Content of attached cd

The cd attached to this thesis includes the following:

- The Java source code for the frontend described in chapter 5 - can be found in the directory `Java_frontend_source_code`.
- The Java source code for the examples in chapters 5 and 6 - can be found in the directory `Examples_Java_source_code`.
- The timed automata models for the examples in chapters 5 and 6 as XML files. These can be used as input to the UPPAAL system. They have been tested on UPPAAL version 4.0.2 - can be found in the directory `Examples_UPPAAL_models`.
- Electronic versions of this thesis in PDF and Postscript formats - can be found in the directory `Thesis_electronic_versions`.

Bibliography

[1] Rajeev Alur and David L. Dill Gerd. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 2004.

[2] Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Times - a tool for modelling and implementation of embedded systems. *Lecture Notes in Computer Science*, 2280:460–464, 2002.

[3] Gerd Behrmann, Alexandre David, and Kim G. Larsen. A tutorial on uppaal. *Lecture Notes in Computer Science*, 3185:200–236, 2004.

[4] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[5] Elena Fersman, Paul Pettersson, and Wang Yi. Timed automata with asynchronous processes: Schedulability and decidability. In J-P Katoen and P. Stevens, editors, *Tools and Algorithms for the Construction and Analysis of Systems. 8th International Conference, TACAS 2002*, volume 2280, pages 67–82. Springer-Verlag, April 2002.

[6] Franco Fummi, Stefano Martini, Giovanni Perbellini, Massimo Poncino, Fabio Ricciato, and Maura Turolla. Heterogeneous Co-Simulation of Networked Embedded Systems. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE'04)*. IEEE, Febuary 2004.

[7] Fabiano Hessel, Vitor M.da Rosa, Igor M.Reis, Ricardo Planner, César A.M.Marcon, and Altamiro A.Susin. Abstract RTOS Modeling for Embedded Systems. In *IEEE International Workshop on Rapid System Prototyping*, 2004.

[8] Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, October 1997.

[9] M. Loghi, F. Angiolini, D. Bertozzi, L. Benini, and R. Zafalon. Analyzing On-Chip Communication in an MPSoC Environment. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE'04)*, pages 752–757. IEEE, Febuary 2004.

[10] J. Madsen, K. Virk, and M. J. Gonzalez. A systemc-based abstract real-time operating system model for multiprocessor system-on-chip. In *Multiprocessor System-on-Chip*. Morgan Kaufmann, 2004.

[11] S. Mahadevan, M. Storgaard, J. Madsen, and K. M. Virk. Arts: A system-level framework for modeling mpsoc components and analysis of their causality. In *13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE Computer Society, sep 2005.

[12] R. Le Moigne, O. Pasquier, and J-P. Calvez. A Generic RTOS Model for Real-time Systems Simulation with SystemC. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE'04)*, page 30082. IEEE Computer Society, 2004.

[13] Sudeep Pasricha, Nikil Dutt, and Mohamed Ben-Romdhane. Extending the Transaction Level Modeling Approach for Fast Communication Architecture Exploration. In *Proceedings of the 38th Design Automation Conference (DAC'04)*, pages 113–118. ACM, 2004.

[14] P. Pop, P. Eles, and Z. Peng. Bus Access Optimization for Distributed Embedded Systems Based on Schedulability Analysis. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE'00)*. IEEE Computer Society, 2000.

[15] Kai Richter, Marek Jersak, and Rolf Ernst. A Formal Approach to MpSoC Performance Verification. *IEEE Computer*, 36(April):60–67, 2003.

[16] J. Sun and J. Liu. Synchronization protocols in distributed real-time systems. In *16th International Conference on Distributed Computing Systems*. IEEE Computer Society, May 1996.

[17] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. Real-time calculus for scheduling hard real-time systems. In *International Symposium on Circuits and Systems ISCAS 2000*, volume 4, pages 101–104, Geneva, Switzerland, March 2000.