# Scheduling shared memory multicore architectures in AUTOFOCUS 3 using Satisfiability Modulo Theories

Sebastian Voss
fortiss GmbH
Guerickestr. 25
80805 Munich, Germany
Email: voss@fortiss.org

Bernhard Schätz
fortiss GmbH
Guerickestr. 25
80805 Munich, Germany
Email: schaetz@fortiss.org

***Abstract*—This paper presents an approach to task and message scheduling for multicore architectures using a shared memory. A shared memory architecture is used in most of the current multicore systems. The presented approach is integrated in the AUTOFOCUS 3 toolchain that allows to automatically compute schedules for this architecture to fulfill certain system requirements. AUTOFOCUS 3 models are based on a formally defined system model using explicit data-flow and discrete-time semantics.**

**Our approach relies on a symbolic encoding scheme, based on a scheduling model, that we generate out of AUTOFOCUS 3 system architectures. This encoding scheme relies on the AUTOFOCUS 3 semantics of strong- and weak-causal components and enables to generate schedules, that fulfill certain system properties. In this paper we focus on finding optimized schedules with respect to the global discrete time base. This paper provides a formalization that describes the scheduling problem as a satisfiability problem using boolean formulas and linear arithmetical constraints. A state-of-the-art satisfiability modulo theories (SMT) solver is then used to compute these schedules. This paper demonstrates that state-of-the art satisfiability modulo theories solver can be used to to compute a schedule that fulfills certain system requirements and meet the challenges of providing both a convenient modeling language and the performance to solve industrialized-sized design problems.**

## I. INTRODUCTION

In embedded systems, applications consist of a number of tasks with precedence constraints, that is, constraints specifying their execution ordering. These tasks are allocated to a set of computing resources (cores) and communicate via messages passing over a shared communication resource. In most of the current multicore systems, this shared communication resource is implemented as a shared memory. Shared memory architectures are convenient in their use, however separation must be guaranteed, as the shared memory can be used by multiple applications, allocated to different cores. In this paper, we assume that separation is guaranteed by the use of, for instance, a local memory protection unit (MPU). The MPU prevents an erroneous application to modify the memory content of another one, as MPUs are used to restrict access to predefined memory regions to guarantee spatial separation.

For such a system composed of a number of tasks communicating over a shared memory, effective configuration is needed to guarantee functional and non-functional system requirements [1]. Effective configuration can be provided by a scheduling policy that provides a suitable off-line schedule, namely an execution ordering of task and messages, corresponding to the given deployment (the allocation of tasks to cores).

Therefore, a scheduling policy has to take into account not only the constraints imposed by the applications (e.g. precedence relations and causality infomation of the application tasks) but also the characteristics and efficient usage of the underlying communication and computation system. Thus, in this paper, problems of task scheduling and message scheduling are regarded in an integrated way, generating schedules that are optimized with respect to the AUTOFOCUS 3 global discrete time base.

AUTOFOCUS 3 is the third version of the AUTOFOCUS tool developed at the Software & Systems Engineering group at the Technische Universität München (`http://af3.fortiss.org`). AUTOFOCUS 3 allows modeling and validating concurrent, reactive, distributed, timed systems on the basis of clearly defined formal semantics [2]. It provides a graphic user interface to specify embedded systems in different layers of abstraction while supporting different views on the system model.

### A. Related Work

Task scheduling for embedded systems has been intensively studied in literature. Well-known preemptive and non-preemptive task scheduling approaches do not take into consideration bus-related communication aspects. Specific issues, such as communication protocols, assignment of slots to messages, etc. are not addressed. These aspects are, however, highly important for guaranteeing predictable system behavior in the context of safety-critical distributed applications.

In this paper we address a scheduling policy for TDMA-based applications that consider both task and message scheduling. Existing task allocation approaches can be roughly clas-

sified into two branches, namely static (offline) and dynamic (online) approaches. The static scheduling approach [3], [4] has been studied in past decades quite intensively in various groups [5]–[7]. The usability of SAT solvers for scheduling synthesis of distributed system has been presented by [8]. In that approach tasks are scheduled by a preemptive, fixed-priority algorithm: Deadline Monotonic with preemption. The approach presented here is based on the transformation of the scheduling problem into nonlinear integer optimization problems, solved by an appropriate propositional SAT checker.

Combined task and message scheduling is considered in several other approaches. The approach advocated by Pree and Farcas [9] introduces algorithms for automatic schedule generation for task and message scheduling. This approach is based on the so-called Logical Execution Time (LET) abstraction, which is harnessed by languages such as Giotto [10] and the Timing Definition Language. Different techniques are employed for task and message scheduling, namely, Earliest Deadline First (EDF) for task scheduling and an heuristic algorithm, adapted from Reverse EDF for message scheduling, respectively.

Dynamic (online) scheduling strategies for shared memory strategies are investigated in [11]. The authors present how to distribute the task assignment function to the processors by having idle processors allocate work to themselves from a shared queue. A run-time spatial mapping and demonstration of streaming applications on heterogeneous MPSoCs at run-time is presented in [12]. Due to the high complexity of the problem, pure runtime strategies mostly rely on fast heuristics

The general problem of task systems with precedence constraints on multiprocessor platforms has been intensively studied in literature. [13] provides a feasibility analysis that determines (prior to system execution-time) whether a specified collection of hard-real-time jobs executed on a processing platform can meet all deadlines.

### B. Organization

The paper is structured as follows. Section II provides basic terms and notations, formalizes the task and message scheduling problem and gives an overview about satisfiability modulo theories. The goal of this paper is reflected in section IV. Section III provides on overview of the AUTOFOCUS 3 tool. In section V we describe how we abstract to generate a scheduling out of AUTOFOCUS 3 architectures. The novel approach is presented, including its symbolic encoding in section VI. The industrial use case and its formalization is described in section VII. Here, the obtained results of the presented approach are described as well. We conclude in Section VIII.

## II. BASIC NOTATIONS AND DEFINITIONS

In order to make this paper as self-contained as possible, we provide some notations and definitions in the following.

### A. Scheduling Model for Multicore Systems

A system may consist of several components for providing various functions. These functionalities can be described as computational activities, called *tasks*. We define $T = \{t_0, t_1, ..., t_n\}$ as a set of tasks. As tasks generally cannot be executed in arbitrary order, the dependency of tasks need to be described with respect to their precedence relations. These relations define the execution ordering and are captured within a strongly directed labeled graph, called *precedence graph* $\mathcal{G} = \{V, E\}$, where each vertex $V$ represents a task $t$, while $E \subseteq V \times 2^M \times V$ represents a set of edges between the tasks. (cf. figure 3). $M$ represents the set of messages: $M = \{m_0, m_1, ..., m_o\}$, that are input/output messages of the tasks. Furthermore, we split up $E$ by using two different functions: $\tau$ and $\rho$, with $\tau : T \rightarrow 2^M$, and $\rho : M \rightarrow T$, where $\tau$ describes all sender tasks $t_{send} \in T$ that trigger a set of messages $m \in M$, and $\rho$ describes for each message $m \in M$ the corresponding receiving task $t_{rec} \in T$. Thus, sender task $t_{send}$ and receiver task $t_{rec}$ correspond to a precedence relation defined in $\mathcal{G}$.

A computation resource may execute a set of concurrent tasks, that is, tasks that can overlap in time. These computation resources are called *cores*. Let $C = \{c_0, c_1, ..., c_m\}$ be a set of cores. Furthermore, let $\eta : C \rightarrow 2^T$ be a function that assigns to every node a set of tasks running on it. A bus $B$ is specified as well. For simplicity reasons, we focus on a single communication resource, that can be used by all computing resources.

Furthermore, we specify an off-chip shared memory resource ($MEM$) as it is common in most multicore systems. This memory can be used by different applications located on different cores, and enables message passing between different cores of $C$.

Using the definitions and notations above, we are able to define a scheduling system as a tuple $S = \langle T, M, B, C, MEM, \eta, \rho, \tau \rangle$, where $T$ is the set of tasks, $M$ is the set of messages, $C$ is the set of cores, $B$ comprises to the bus in the system and $MEM$ is the shared memory resource. Precedence relations imposed by the precedence graph $\mathcal{G}$ are defined by the allocations $\tau$ and $\rho$. Allocations concerning deployment information of tasks to nodes are defined by $\eta$. In this paper, we assume that the elements of $S$ corresponds to the logical execution time (LET), which is a timed model, where all computational activities and communications take logically a fixed amount of time, regardless of whether they actually need less time to execute.

### B. Satisfiability Modulo Theories

Satisfiability Modulo Theories (SMT) enables checking the satisfiability of logical formulas over one or more theories. SMT combines the boolean satisfiability with other background theories, such as, linear arithmetic, arrays, uninterpreted functions, etc. (cf. [14]). Thus, the well-known constraint satisfaction problem of propositional satisfiability SAT, where the goal is to decide whether a formula over boolean variables can be made `true` by choosing `true/false` values for its variables, is extended by more expressive logics such as first-order logic. First-order logic formulas consists of logical connectivities, variables, quantifiers, functions and

predicate symbols. In SMT, interpretations of some symbols are constrained by a background theory (e.g. linear arithmetics, etc.). SMT provides a *model* as a solution. This model consists of interpretations for the variable, function and predicate symbols that makes the formula `true`. Further information on satisfiability modulo theories can be found in [14].

## III. The case tool AutoFOCUS 3

This section provides information about the case tool AUTOFOCUS 3 . First, we describe the semantics (e.g. model of computation, component semantics, etc.), then we explain the layers of abstraction.

### A. AutoFOCUS 3 *semantics*

AUTOFOCUS 3 uses a message-based, discrete-time communication scheme as its core semantic model. As AUTOFOCUS 3 is designed using networks of components, these components communicate via messages. Messages are time-stamped with respect to a global, discrete time base. This computational model supports a high degree of modularity by making component interfaces complete and explicit. The component-based system architecture corresponds to the previous named layer of logical architecture. The discrete time base abstracts from implementation details such as detailed timing or communication mechanisms because the usage of timing information below that chosen granularity of observable discrete clock ticks is avoided. The message-based time-synchronous communication model caters for both periodic and sporadic communication behavior.

Furthermore, AUTOFOCUS 3 provides different component semantics with respect to timing: the notion of *strong* and *weak* causality. Assuming a global, discrete notion of time, meaning that time advances in discrete logical units and every component has the exactly same time information of this global time tick, the systems start at the logical time 0 and all components start synchronously in some initial state. At the beginning of each such computation step, each atomic component reads its input values and produces the output depending on its specified behavior (automation specification). Strong causality means that the output of each component becomes visible to the specified communication partner (another component or the environment) at the beginning of the next global tick. This delay of one logical time unit is meant by the strong causality assumption. Weak causality means that instantaneous reaction on a given input without a time delay of one global tick. The current output may therefore depend on the current input.

### B. AutoFOCUS 3 *layers of abstraction*

An AUTOFOCUS 3 system model is divided into several models, that provides different layers of abstractions. It provides a graphic user interface to specify embedded systems. The AUTOFOCUS 3 tool provides several types of view for the layers, e.g. for the logical layers a data definition view, a system structure view and behavior view are provided, as described in [15]. Other supporting views (e.g. system structure diagrams (SSD), state transition diagram (STD), etc.) are provided as well .

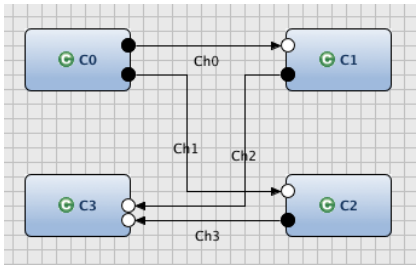


Fig. 1. Graphical representation of AUTOFOCUS 3 logical architecture

From a deployment point of view, there are two different kinds of system architectures: the logical architecture (LA) and the technical architecture (TA).

The logical architecture of a system (compare figure 1) is defined by means of logical components communicating via defined communication paths. Each component exposes defined input and output interfaces to its environment, either to other components or to the system environment. These interfaces are specified via a set of typed, input (empty circles in figure 1) or output ports (filled circles in figure 1). Composition of components is defined by introducing channels. A channel is a communication path between two ports of some components, namely defining sender / receiver relations. AUTOFOCUS 3 is characterized by a message-based, discrete-time communication scheme as its core semantic model. Thus, it does cater for both periodic and sporadic communication as required for a mixed modeling of time-triggered and event-triggered behavior.

Figure 1 illustrates the logical architecture of a simple example that is used through this paper to demonstrate the presented approach. We consider a set of AUTOFOCUS 3 components $Comp = \{C_0, C_1, C_2, C_3\}$ and a set of channels $Ch = \{Ch_0, Ch_1, Ch_2, Ch_3\}$ that connect the number of output and input ports in order to describe the communication paths of the given architecture.

The technical architecture describes a hardware topology that is composed of hardware components (compare figure 2), namely cores, which in turn may consist of hardware ports (sensors or actuators), busses and a shared memory. A deployment logic is then used to map components from the logical to the technical architecture. We use these information in order to generate a precedence graph. In the next subsection we describe how we abstract that precedence graph.

## IV. Objective

Our objective is to find a task schedule which incorporates a message schedule that finds the shortest logical tick duration while considering reliable and predictable communication based on given precedence graph and AUTOFOCUS 3 semantics of strong and weak - causal components. Two aspects are considered:

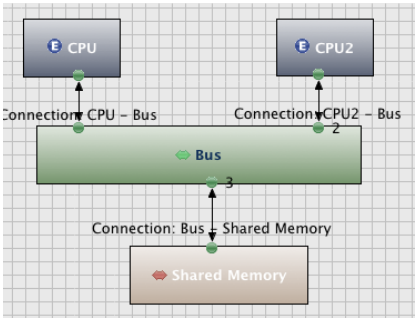


Fig. 2. Graphical representation of AUTOFOCUS 3 technical architecture

The task and message scheduling problem can be described as follows:

*The scheduling problem consists of determining the starting time for all tasks in $T$ and all messages in $M$, using a non-preemptive schedule, such that the complete schedule does not exceed a given time bound (logical tick). This schedule calculates for each task $t_i \in T$ the tuple*

$\gamma_i = \langle s(t_i), \{s(m_1), \ldots, s(m_k)\}\rangle$, with $\{m_1, \ldots, m\} = \tau(t_i)$.

*, where $s(t_i)$ corresponds to the starting time of $t_i \in T$, and $s(m_k)$ corresponds to the starting times of all messages $m \in M$ that are defined in $\tau(t_i)$.*

We focus on the generation of schedules that comply to a certain requirements. We present an approach that calculates a schedule $\gamma$ that fulfills the requirement of optimizing the global discrete time base. We specify this time base duration as a length $l$ = max $(f(t_i) - s(t_j))$, $\forall t \in T$. $f(t_i)$ represents the finishing time of a task $t_i$ and s($t_j$) is the starting time of $t_j$.

In this paper two things should be demonstrated. First, this paper demonstrates how state-of-the-art SMT solvers (e.g. YICES, CVC) can be used to generate these schedules that incorporate task scheduling at system level and message scheduling at communication level. This includes the order of tasks, defined by the precedence graph, needs to be considered for defining a proper task schedule (sequence and starting time of the different tasks based on their weak or strong causal semantic), as well as a message schedule has to be defined, which assigns message communication information, e.g. starting time, delay of transfer (e.g. time to write the contained information into the memory). Therefore, our goal is to find a solution for the given scheduling problem by using SMT capabilities. In the following we describe the translation of this problem with boolean formulas and linear arithmetic constraints.

Secondly, we demonstrate how this approach extends the existing AUTOFOCUS 3 model, by providing a further degree of system description to the model. Existing AUTOFOCUS 3 models provide a reduced degree of complexity, because the discrete time base abstracts from implementation details such as detailed timing or communication mechanisms. Thus, this approach provides information below the chosen granularity of observable discrete clock ticks. These timing information includes all information related to the given scheduling problem, e.g. the ordering of tasks and messages, starting times of tasks and messages, as well as information concerning shared memory access (read/write) times. Therefore, the calculation of an AUTOFOCUS 3 ”tick” becomes possible by the presented approach.

## V. GENERATING THE SCHEDULING MODEL

This section describes how we generate a scheduling system model $S$, as defined in section II-A. In order to calculate a scheduling model, we gather information from the logical architecture as well as the technical architecture of AUTOFOCUS 3 models.

Based on the logical and technical architecture of an AUTOFOCUS 3 model, we are able to abstract the scheduling model that is used for the given scheduling approach. The logical architecture provides a set of components that, in general, represents a set of tasks and a set of channels that represent messages of a given precedence graph. The technical architecture provides the hardware resources (e.g. number of cores, busses and the shared memory). Using these informations, we generate a scheduling model for the simple example presented in figure 1 as a set of tasks $T = \{t_0, t_1, t_2, t_3\}$, a set of messages $M = \{m_0, m_1, m_2, m_3\}$ and a set of cores $C = \{c_0, c_1\}$. The allocation of task to cores can be described by $\eta(c_0) = \{t_0, t_3\}$ and $\eta(c_1) = \{t_1, t_2\}$

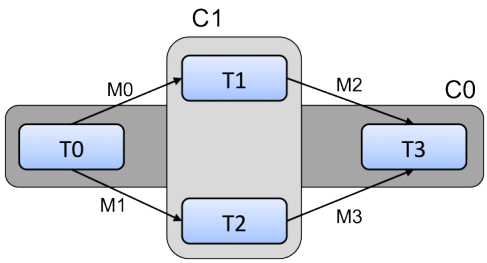


Fig. 3. Graphical Visualization of a precedence graph $\mathcal{G}$

Furthermore, the precedence relations need to be specified. This can be done as follows: $\tau(t_0) = \{m_0, m_1\}$, $\tau(t_1) = \{m_2\}$, $\tau(t_2) = \{m_3\}$, $\tau(t_3) = \{\}$ and $\rho(m_0) = \{t_1\}$, $\rho(m_1) = \{t_2\}$, $\rho(m_2) = \{t_3\}$ and $\rho(m_3) = \{t_3\}$.

The extracted scheduling model can be graphically represented by an extended precedence graph $\mathcal{G}$ (as defined in section II-A) as shown in figure 3.

## VI. SMT - BASED MULTICORE TASK AND MESSAGE SCHEDULING

This paper presents a novel approach to task and message scheduling by using a formalization that describes the scheduling problem as a satisfiability problem using boolean formulas and linear arithmetical constraints. We demonstrate that efficient SMT solvers can be used for scheduling synthesis.

### A. SMT Solver YICES

YICES [16] is an efficient SMT solver developed at SRI International (`http://yices.csl.sri.com`). It supports a combination of first-order theories, such as arithmetics, uninterpreted functions with equality, bit vectors, arrays, recursive datatypes, and more. YICES is able to solve classical SMT problems, namely it decides the satisfiability of propositionally complex formulas in such theories. Further information concerning YICES architecture and algorithms can be found in [17].

### B. Translation to YICES

We propose to solve the scheduling problem, as formulated in section II-A by using an efficient SMT solver. Therefore, we need to encode the scheduling problem as a decision problem using boolean formulas with linear arithmetic constraints in order to check the validity. By finding a valid solution, we are then able to generate schedules that comply to the requirements of an optimized global discrete time base $l$, as defined in section IV. With respect to this goal of finding an optimized task and message schedule based on AUTOFOCUS 3 semantic, we implement a binary search fining the shortest latency possible.

*1) Assumptions:* As explained in section IV, our objective is to demonstrate how state-of-the-art SMT solvers can be used to generate task and message schedules. Thus, this approach aims to calculate starting times for all tasks $t \in T$ and messages $m \in M$. The following assumptions are used in this paper:

- The precedence graph is defined a priori. The assignments ($\eta : N \rightarrow 2^T$, $\tau : T \rightarrow 2^M$ and $\rho : M \rightarrow T$) are defined as well. Preemption of tasks is not considered.
- For simplification purposes, we expect the computing resources (cores) in the system to be identical concerning computation speed; i.e., the computation time of a task is the same on all nodes.
- As messages are input respectively outputs of a certain task (corresponding to the given precedence graph $\mathcal{G}$), the precedence relations have to be guaranteed, according the their causality.
- As each message $m \in M$ is transferred via a write and read operation in and out of the shared memory $MEM$, we distinguish between a write and read part for each message $m \in M$.
- Furthermore, for simplification purposes, a single communication bus $B$ and a single shared memory $MEM$ are assumed.
- The time which is estimated as communication duration for each write and read message $m \in M$ corresponds to the time for transmitting it over the bus and writing it into the shared memory. The read and write operations are accumulated as task computation time.

*2) Definitions:* The given precedence graph $\mathcal{G}$ combined with a deployment, as specified in section II-A, comprises several elements: a set of cores, a set of tasks and a set of messages. Thus, we begin by defining type declarations for these precedence graph elements in YICES.

Definition 1 (Cores): The set of cores is defined as $C = \{c_1, \ldots, c_s\}$. All cores $c \in C$ are specified in YICES input language as follows:

```
define-type cores (subrange 1 s),
```

where $s$ complies to the number of cores in $C$.

Definition 2 (Tasks): To specify a task type, we define four properties in a dedicated data structure that is defined as a record type in YICES input language, as follows:

```
(define-type task (record
start_time :: nat
computation_time :: nat
```

```
complete_time :: nat
core :: cores))
```

The defined task record stores the parameter information of a single task $t_i$. The variable $start\_time$ stores the starting time of a task. The $computation\_time$ stores the given computation duration of a task. The $complete\_time$ stores the finishing time of a task. The variable $node$ represents the node a task is allocated to. The $computation\_time$ and $core$ are predefined properties according to the first assumption.

Definition 3 (Messages): We specify a message type by using a message record (comparable to Definition 2) that stores the parameter information of a single message $m_i$. The variable $start\_time$ stores the starting time of a message. The $communication\_duration$ stores the given transmission duration and the $complete\_time$ stores the finishing time of a message.

```
(define-type message (record
start_time :: nat
communication_duration :: nat
complete_time :: nat))
```

*3) Assertions:*

Assertion 1 (Executing Parameters Constraints): The computation time of tasks and the communication duration of messages are defined a priori and can be described in a formal way as follows:

$\models$ $t_i$.computation_time = v
$\models$ $m_i$.communication_duration = v,

where v is the given value. In YICES they are specified as follows:

```
(assert(=(select(tasks i)computation_time) v))
(assert(=(select(messages i)communication
_duration) v))
```

Assertion 2 (Task Allocation Constraints): This assertion guarantees that the computation times of all tasks that are allocated to the same computing resource (node), are disjoint, meaning there is only one task at most that is currently using the recourse at any point in time. This is described in a formal way as follows:

$\not\models \exists$ t (t $\in$ Time) (
$t_i$.start_time $\leq$ t $<$ $t_i$.complete_time $\wedge$
$t_j$.start_time $<$ t $\leq$ $t_j$.complete_time $\wedge$
$t_i$.node = $t_j$.node $\wedge$ $t_i \neq t_j$)

It is specified in YICES input language as follows:

```
(assert (not (exists (t::nat)
(and (>= t (select (tasks i) start_time))
(< t (select (tasks i) complete_time))
(> t (select (tasks j) start_time))
(<= t (select (tasks j) complete_time))
(= (select (tasks i) node)
(select (tasks j) node)) (/= i j))))
```

Assertion 3 (Precedence Graph Constraints): According to the third assumption all precedence relations defined in $\tau(t_{send}) = \{m_{i\_write}\}$ and $\rho(m_{i\_read}) = \{t_{rec}\}$ should be met.

Here, we need to make two distinctions:

One corresponding to the causality of the sender task, as a sender task ($t_{send}$) is derived from a weak-causal component of AUTOFOCUS 3 logical architecture, the complete time of the sender task ($t_{send}$) and the start time of $write$ - part of message message ($m_i$) (indicated as $m_{i\_write}$) should be equal and the complete time of the $read$ - part of this message should be less or equal to the start time of the receiver task ($t_{rec}$). Note that $write$ and $read$ part of the message can be timely separated, as this is one of the characteristics using shared memory systems.

$\models$ ($m_{i\_write}$.start_time = $t_{send}$.complete_time) $\wedge$
($m_{i\_read}$.complete_time $\leq$ $t_{rec}$.start_time),

where message $m_{i\_write}, m_{i\_read} \in M$ and $t_{send}, t_{rec} \in T$. It is specified in YICES input language as follow:

```
(assert (and (= (select (message i_write)
start_time)
(select (tasks send) complete_time))
(<= (select (messages i_read) complete_time)
(select (tasks rec) start_time))))
```

where send and rec are id's for the actual sender and receiver tasks and i is the id for a corresponding message.

In case a sender task ($t_{send}$) is derived from a strong-causal component of AUTOFOCUS 3 logical architecture, the complete time of the sender task ($t_{send}$) should be greater or equal to the start time of the message ($m_{i\_write}$) .

$\models$ ($m_{i\_write}$.start_time >= $t_{send}$.complete_time)

where message $m_{i\_write} \in M$ and $t_{send} \in T$. It is specified in YICES input language as follow:

```
(assert (>= (select (message i_write)
start_time)
(select (tasks send) complete_time)))
```

where send and rec are id's for the actual sender and receiver tasks and i is the id for a corresponding message.

Assertion 4 (Message Allocation Constraint 1): This assertion guarantees the disjoint access of messages to a shared communication resource, meaning that there is only one message at most which can be transmitted and written into the shared memory simultaneously. We describe this in a formal way as follows:

$\not\models \exists$ t (t $\in$ Time) (
$m_i$.start_time $\leq$ t $<$ $m_i$.complete_time $\wedge$
$m_j$.start_time $<$ t $\leq$ $m_j$.complete_time $\wedge$ $m_i \neq m_j$)

This is specified in YICES input language as follows:

```
(assert (not (exists (t::nat) (and (>= t
(select (messages i) start_time))
(< t (select (messages i) complete_time))
(> t (select (messages j) start_time))
(<= t (select (messages j) complete_time)) (/=
i j))))
```

We use this assertion for all kinds of messages (the $write$ and the $read$ part) of a message $m_i \in M$.

Assertion 5 (Message Allocation Constraint 2) We need to distinguish two different cases: In case the sender task $t_{send}$ and the receiver task $t_{rec}$ are allocated to the same computing resource, the complete time is calculated as follows:

$m_i$.complete_time = $m_i$.start_time
iff: $t_{send}$.node = $t_{rec}$.node

In case, sender task $t_{send}$ and receiver task $t_{rec}$ are allocated to different computing resources, the complete time is calculated as follows:

$m_i$.complete_time = $m_i$.start_time+
$m_i$.communication_duration
iff: $t_{send}$.node $\neq$ $t_{rec}$.node

This assertion is also used for all kinds of messages (the $write$ and the $read$ part) of a message $m_i \in M$.

It is specified in YICES input language as follows:

```
(assert (= (select (messages i) complete_time)
(if (= (select (tasks send) node)
(select (tasks rec) node))
(select (messages i) start_time)
(+ (select (messages i) start_time)
(select (messages i) communication_duration))))
```

We can use functions instead of quantifiers when it is necessary. Finally, we specify the given system requirement as a correctness property for the given SMT - based scheduling approach. This is presented in the next section.

### C. (Correctness) Properties

As described in section IV, we specify the global discrete time base as a length $\mid l \mid$. Therefore, we need to define the function min which provides the minimum value and max which provides the maximum value. Using these functions, we are able to calculate the logical tick duration $\mid l \mid$. This is defined in YICES input language as follows:

```
(define duration::(subtype (n::nat) (= n (-
(max (select (tasks 1) complete_time)
(select (tasks 2) complete_time) ...
(select (tasks n) complete_time))
(min (select (tasks 1) start_time)
(select (tasks 2) start_time) ...
(select (tasks n) start_time))
```

Finally, the `check` command checks whether the current logical context is satisfiable or not. If the scheduling problem is satisfiable using the constraints imposed by the given end-to-end system requirement, a solution model is given.

## VII. Industrial Use-Case

In the following we present an automotive use case to demonstrate the usability of the presented approach using a real world example.

### A. Adaptive Cruise Control (ACC) - System

The Adaptive Cruise Control (ACC) is an automotive use case. The ACC automatically adjusts the traveling speed of an automotive vehicle by controlling the acceleration and breaking momentum, based on a driver-defined reference speed, the current speed as well as the distance to a (possibly present) leading vehicle. Figure 4 shows the architecture of the application using AUTOFOCUS 3 , consisting of 5 main components (speed and distance plausibilization; speed- and distance-based control; (de-)acceleration computation). During deployment, these components are mapped to 5 corresponding tasks allocated to 2 cores and connected via an Avalon bus, that is connected to the shared memory (cf. figure 2 in subsection III-B).

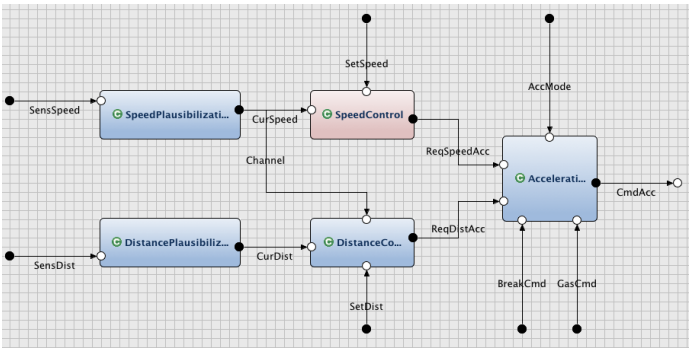


Fig. 4. Automotive Use Case: Adaptive Cruise Control (ACC)

The figure 4 describes the way how the Adaptive Cruise Control system is modeled using AUTOFOCUS 3 . Each component might consists of further components or have state transition diagrams used to specify the behavior (not shown in figure 4).

### B. ACC Schedule Synthesis

The presented approach is demonstrated using the AUTOFOCUS 3 system model in section VII. This model is transformed into a scheduling model represented by an extended precedence graph $\mathcal{G}$ (cf. sectionII-A and V), which is used as a basis for the SMT-based scheduling approach.

We start generating the scheduling model as described in section II-A for the given example as a set of 5 tasks $T = \{t_{SpeedPlausibilization}, t_{DistancePlausibilization}, \cdots, t_{Accerlation}\}$ and a set of 13 messages $M = \{m_{SensSpeed}, \cdots, m_{CmdAcc}\}$ and 2 cores $C = \{c_1, c_2\}$.

We start defining assertions that are related to the application / system under consideration. Therefore, all predefined values, namely the computation time (computation_time) for each task $t \in T$ and the expected values for the communication duration (communication_duration) of all messages $m \in M$ are set.

We use the YICES `assert` command for specifying this:

```
(assert (= (select (task
SpeedPlausibilization) computation_time) 10))
...
```

```
(assert (= (select (m CurSpeed)
communication_duration) 2))
...
```

In a next step, we define constraints that are imposed by the precedence graph $\mathcal{G}$. As tasks and messages cannot be scheduled in an arbitrary order, precedence relations are defined by the functions $\tau$ and $\rho$ (cf. section II-A) are used to guarantee all precedence relations in $\mathcal{G}$ (e.g. $\tau(t_{DistancePlausibilization}) = \{m_{CurSpeed}\}$ and $\rho(m_{CurSpeed}) = \{t_{DistanceControl}\}$).

Furthermore, we specify the constraints that are imposed by the allocation of tasks to cores $\eta$ (e.g. $\eta(c_0) = \{t_{DistancePlausibilization}\}$).

Finally, we specify the given system requirement. As we focus on minimizing the logical tick duration $\mid l \mid$ in this paper, we demand for a schedule with a latency of 100 $\mu s$ using the relation $duration$ (cf. section VI-C).

```
(assert (<= duration 100))
```

### C. Satisfied Solution Model

The function of a SMT solver is to check the satisfiability of logical formulas over one or more theories. The solution model provided by the SMT solver is a valid solution for the given scheduling problem. However, the SMT solver outputs one solution that fulfills the defined constraints. A *valid* solution, a *model*, consists of interpretations for the variables, functions and predicate symbols that makes the formula true. For analysis and demonstration of operation, we invoke YICES on the given Adaptive Cruise Control (ACC) - System.

Solution Model given by YICES SMT - Solver:

```
(= duration 30)
(= (task DistancePlausibilization)
(mk-record
start_time :: 5
computation_time :: 10
core :: C0))
(= (task SpeedPlausibilization)
(mk-record
start_time :: 17
...

(= (message CurSpeed_write
(mk-record
start_time :: 27
communication_duration :: 2
(= (message CurDist
...
```

The solution model comprises all defined precedence graph elements. This includes a solution for the defined task and message scheduling problem. For instance, task $t_{DistancePlausibilization}$ has a calculated $starting\_time$ of 5 $\mu s$. Task $t_{SpeedPlausibilization}$ starts at 17 $\mu s$. In addition to the task, message properties are included as well: Message $m_{CurSpeed\_write}$, for instance, has an allocated starting time of 27 $\mu s$.

Thus, based on the solution model provided, we are able to extract an integrated task and message schedule $\gamma = \{t_i \mapsto \gamma_i, \forall t_i \in T\}$ that is integrated into the AUTOFOCUS 3 system model.

$$\begin{aligned}\gamma = \{&t_{SpeedPlausibilization} \mapsto \langle 17, \{27, 0\}\rangle,\\ &\{t_{DistancePlausibilization} \mapsto \langle 5, \{15, 3\}\rangle,\\ &\{t_{SpeedControl} \mapsto \langle 10, \{\}\rangle,\\ &\{t_{DistanceControl} \mapsto \langle 2\rangle 0, \{\}\\ &\{t_{Acceleration} \mapsto \langle 0, \{\}\rangle\}\end{aligned}$$

The calculated schedule is illustrated in figure 5. As explained in section II, tasks $t_{SpeedPlausibilization}$ and $t_{DistancePlausibilization}$ are allocated to the same node $n_{ECU0}$. The other tasks are allocated to the node $n_{ECU1}$ Hence, the execution ordering needs to be disjoint for that node.

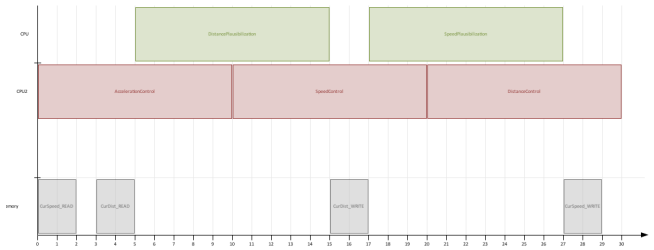

Fig. 5. Optimized Schedule of Active Cruise Control

Thus, the optimized global discrete time base $l$ for the given AUTOFOCUS 3 system model under consideration is given as 30 $\mu s$. YICES SMT Solver needed $0,05$ seconds to calculate this optimized solution for this scheduling problem.

## VIII. CONCLUSION AND FUTURE WORK

In this paper, we have presented an approach to combined task and message scheduling that allows to compute schedules that fulfil certain system requirements. This paper demonstrates that state-of-the-art SMT solvers (we use YICES from SRI) can be used to generate these schedules. Based on a given AUTOFOCUS 3 system model, we calculate a precedence graph as a basis for task and message scheduling. We have described a modeling concept for abstracting the given problem that can be used to calculate a schedule for certain end-to-end latency requirements. Experimental results demonstrate both that a given scheduling problem can be described as a SMT problem and that a state-of-the-art SMT solver meets the challenges of providing the performance to solve these problems.

Future work is focused on an extended version of the given approach that enables to calculate not only schedules fulfilling given (hard) real-time system requirements, but also an optimized mapping (e.g. allocation of tasks to nodes) that fulfills these requirements.

## REFERENCES


[1] M. Paulitsch, H. Rueß, and M. Sorea, "Non-functional avionics requirements," 2008.

[2] A. Bauer, M. Broy, J. Romberg, B. Schätz, P. Braun, U. Freund, N. Mata, R. Sander, and D. Ziegenbein, "Automode - notations, methods, and tools for model-based development of automotive software," in *SAE International*, 2005.

[3] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Norwell, MA, USA: Kluwer Academic Publishers, 1997.

[4] D.-T. Peng, K. G. Shin, and T. F. Abdelzaher, "Assignment and scheduling communicating periodic tasks in distributed real-time systems," *IEEE Transactions on Software Engineering*, vol. 23, pp. 745–758, 1997.

[5] P. Pop, P. Eles, and Z. Peng, "An improved scheduling technique for time-triggered embedded systems," in *EUROMICRO Conference Procededings*, no. 25. ACM, 1999, pp. 303–310.

[6] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and Microprogramming - Euromicro Journal (Special Issue on Parallel Embedded Real-Time Systems)*, vol. 40, pp. 117–134, 1994. [Online]. Available: citeseer.ist.psu.edu/tindell94holistic.html

[7] T. F. Abdelzaher and K. G. Shin, "Combined task and message scheduling in distributed real-time systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 10, no. 11, pp. 1179–1191, 1999.

[8] A. Metzner, M. Fränzle, C. Herde, and I. Stierand, "Scheduling distributed real-time systems by satisfiability checking," in *RTCSA '05*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 409–415.

[9] E. Farcas, C. Farcas, W. Pree, and J. Templ, "Transparent distribution of real-time components based on logical execution time," *SIGPLAN Not.*, vol. 40, no. 7, pp. 31–39, 2005.

[10] T.Henzinger, B.Horowitz, and C. Kirsch, *Giotto: A time-triggered language for embedded programming*. Springer-Verlag, 2001.

[11] B. Hamidzadeh and D. Lilja, "Dynamic scheduling strategies for shared-memory multiprocessors," in *Distributed Computing Systems*, 1996.

[12] P. K. F. Hölzenspies, J. L. Hurink, J. Kuper, and G. J. M. Smit, "Run-time spatial mapping of streaming applications to a heterogeneous multi-processor system-on-chip (mpsoc)," in *Proceedings of the conference on Design, automation and test in Europe*, ser. DATE '08. New York, NY, USA: ACM, 2008, pp. 212–217. [Online]. Available: http://doi.acm.org/10.1145/1403375.1403427

[13] N. Fisher and S. Baruah, "The feasibility of general task systems with precedence constraints on multiprocessor platforms," *Real-Time Syst.*, vol. 41, pp. 1–26, January 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1485069.1485082

[14] L. de Moura and N.Bjoerner, "Satisfiability modulo theories: An appetizer," in *SBMF*, 2009, pp. 23–36.

[15] H. Lötzbeyer and A. Pretschner, "Autofocus on constraint logic programming," in *Logic Programming and Software Engineering (LPSE)*, 2000.

[16] B. Dutertre and L. de Moura, "The yices smt solver," Computer Science Laboratory, SRI International, Tech. Rep.

[17] ——, "Fast linear-arithmetic solver for dpll(t)," in *Proc. 18th Computer-Aided Verification conference*, ser. LNCS, vol. 4144. Springer-Verlag, 2006, pp. 81–94.