Process network models for embedded system design based on the real-time BIP execution engine [*]

Fotios Gioulekas
Department of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
gioulekas@csd.auth.gr

Peter Poplavko
Mentor®. A Siemens Business
110, rue Blaise Pascal
Inovalle Montbonnot
38334 ST ISMIER CEDEX
France
ppoplavko@gmail.com

Panagiotis Katsaros
Department of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
Information Technologies Institute
CERTH, Thessaloniki, Greece
katsaros@csd.auth.gr

Pedro Palomo
Deimos-Space S.L.U
Madrid
Spain
pedro.palomo@deimos-space.com

Existing model-based processes for embedded real-time systems support the analysis of various non-functional properties, most notably schedulability, through model checking, simulation or other means. The analysis results are then used for modifying the system's design, so that the expected properties are satisfied. A rigorous model-based design flow differs in that it aims at a system implementation derived from high-level models by applying a sequence of semantics-preserving transformations. Properties established at any design step are preserved throughout the subsequent steps including the executable implementation. We introduce such a design flow using a process network model of computation for application design at a high level, which combines streaming and reactive control processing with task parallelism. The schedulability of the so-called FPPNs (Fixed Priority Process Networks) is well-studied and various solutions have been presented. This article focuses on the design flow's steps for deriving executable implementations on the BIP (Behavior - Interaction - Priority) runtime environment. FPPNs are designed using the TASTE toolset, a convenient architecture description interface. In this way, the developers do not program explicitly low-level real-time OS services and the schedulability properties are guaranteed throughout the design steps by construction. The approach has been validated on the design of a real spacecraft on-board application that has been scheduled for execution on an industrial multicore platform.

1 Introduction

The model-based design of embedded real-time systems takes place by modeling the system components and their interactions from the early design stages. An adequate design model includes the application behavior with its partitioning into hardware-software components, the overall system architecture, and the mapping of the application to the system's architecture. Such a high-level model enables the analysis, the early verification of design, and estimations on the system's performance [3].

The related works are roughly classified into two main classes of approaches. Architecture-centric model-based designs are based on an architecture description, whereas via model transformations the system's non-functional properties are analyzed with appropriate tools [11]. Schedulability depends on

[*]The research leading to these results has received funding from the European Space Agency project MoSaTT-CMP, Contract No. 4000111814/14/NL/MH

Submitted to:
MeTRiD 2018

© F. Gioulekas, P. Poplavko, P. Katsaros & P. Palomo
This work is licensed under the
Creative Commons Attribution License.

certain assumptions for the temporal and concurrency properties of computations, communication and their synchronization (e.g. cyclic executive with time-triggered activation, priority-based pre-emption), which render a model statically analyzable. On the other hand, the synchronous approach [9] is suitable for the formal design and verification of reactive systems (e.g. flight control systems) that react to stimuli from the environment within strict time bounds. In synchronous reactive languages, the program reacts in a sequence of logical clock ticks and computations within a tick are instantaneous. In this setting, programs are amenable to formal verification [10] and code generation for embedded platforms [18].

The process network models of computation are mainly used in streaming signal processing and provide a means to cope with the complexity of parallel programming. They dictate a decomposition of behavior into pieces by defining the relationships between these pieces, while enabling the analysis of non-functional properties. Programs are written in the form of directed graphs with nodes for their functions and arcs for the data flows between functions. Such programs can exploit concurrency when they are deployed to parallel hardware, while their functions can be statically scheduled.

In [7], we introduced the formal semantics of a new process network model that combines streaming and reactive control processing with task parallelism. The so-called FPPNs (Fixed Priority Process Networks) enable the design of applications that react to environment stimuli through the definition of communicating tasks that are programmed independently from the execution platform. Task activations depend on a combination of data availability (similar to streaming applications) and complex (non-periodic) arrival patterns. A noteworthy characteristic of FPPN-based designs is their functional determinism, i.e. the fact that for a given test stimuli we expect a deterministic output response. This feature renders applications amenable to testing, as opposed to applications programmed using low-level real-time OS services, whose outputs may depend on their timing behavior. The schedulability properties of FPPNs and the first static scheduling solutions were presented in [16], whereas their applicability in mixed-criticality systems [14] and in systems with shared resource interference [6] have been also studied.

The FPPN semantics is defined in [7] by compilation into RT-BIP [2], an executable formal language for modeling networks of connected timed automata. In present paper, we utilize the results from [7], [16] and [14] to propose a design flow based on the principles of rigorous system design [19]. Such a flow differs from other model-based approaches in that a sequence of semantics-preserving steps allows deriving an executable implementation of an FPPN-based application design. This is achieved by embedding the functional code into the FPPN design model through the high-level architecture description interface of the TASTE toolset [13], whose front-end tools have been amended to capture FPPN-compliant models. The schedulability is established by static analysis of the high-level FPPN, it is preserved throughout the subsequent model transformation steps and it is eventually guaranteed by construction by the derived implementation for the real-time BIP runtime environment that supports parallel execution of BIP components using POSIX threads. This means that the developers can reason in terms of high-level schedulabity concepts (e.g. tasks, priorities, deadlines, offsets etc.) and they do not need to explicitly program low-level real-time OS services (e.g. for task management, inter-task communication, memory allocation, scheduling etc.), thus retaining the predictability advantage of a statically analyzable design.

We experimented on the rigorous design of a Guidance Navigation & Control (GNC) on-board spacecraft application that was ported onto ESA's Next Generation Microprocessor (NGMP), more specifically the quad-core LEON4FT processor [1]. The effectiveness of our approach is shown through measurements on execution traces, which also provide valuable insight for analyzing the scheduling bottlenecks.

In Section 3, we introduce the basics of the FPPN model of computation and we then show the GNC FPPN model captured using the front-end tools of TASTE. Section 4 discusses in detail the design flow steps. Section 5 elaborates the steps of schedulability analysis and code generation, as well as the details of their application on the GNC case study. Section 6 presents the measurements from the execution of the scheduled GNC FPPN C++ programs on top of the BIP runtime environment that was ported onto the LEON4FT processor. The paper concludes with a critical overview of the achievements, a discussion on the gained insight and the future research prospects.

2 Related model-based approaches for real-time embedded systems

The entry point of any model-based design approach is an adequate language, which allows avoiding the premature commitment to specific execution/interaction semantics or implementation choices. In particular, the design of time-critical applications is a challenging procedure, which is usually based on evolutionary prototype building [11]. As the design cycle evolves, modeling formalisms are expected to support refinement, the setting of system attributes and analysis. The non-functional properties of the system are analyzed e.g. by model checking, simulation or other methods, and the results are taken into account in system design. In [3], the authors discuss and address two challenges related to the transformations of the model used for design into models used for analysis: the validity of the transformations and how to take the analysis result into account in system design.

Other related works [17] emphasize the theory of model transformations and the incorporation of suitable models of computation, however, by often ignoring schedulability aspects. Model-based System-Engineering environments like Ptolemy II and PeaCE support a plethora of models of computation as a means for the refinement-based design of (multi-core) real-time embedded systems [8, 12, 4]. To the best of our knowledge, this work presents the first model-based approach grounded on the principles of timing-aware rigorous design [2] and task schedulability on multiple cores.

3 FPPN process networks and their design with the TASTE toolset

A process network model defines a partial order, in which a relative sequence of concurrent executions is specified only for a sequence of events, handled based on causality and inherent dependencies. While the engineer focuses on the key aspects of the application's design, in the physical implementation of every event it is somehow attached to a real-time instant, thus implying a natural total order for the system's execution. The engineers do not need to know the model's formal semantics; they just have to follow composition rules, which establish the dependencies between processes in the form of data and/or control flow.

In [7], the authors proposed the FPPNs, a process network model that combines streaming and reactive control processing. An instance of FPPN is composed of three main entities: *Processes*, *Data Channels* and *Event Generators*. A *Process* represents a software subroutine, i.e. functional code that operates with internal variables and input/output channels connected to it through ports. Every process is mapped to an event generator, which determines whether the process is periodic or sporadic. Periodic and sporadic processes are a generalization of classical periodic and sporadic tasks to communication via channels. Processes are assigned *functional priorities*, which define a relation between processes to ensure deterministic communication. An invocation of a process is referred to as a *job*. Like the real-time jobs, the subroutine should have a bounded execution time subject to WCET (worst-case execution time) analysis.

An FPPN is formally defined by two directed graphs. The first one is a (possibly cyclic) graph (P, C), whose nodes P are processes and edges C are channels for pairs of communicating processes that define a dataflow direction, i.e. from the writer to the reader (there are also external channels interacting with the environment). The second graph (P, FP) is the functional priority directed acyclic graph (DAG) with edges defining a functional priority relation between processes. This part of FPPN's definition ensures its functional determinism, i.e. that the outputs calculated by FPPN depend only on the event invocation times and the input data sequences, but not on the scheduling.

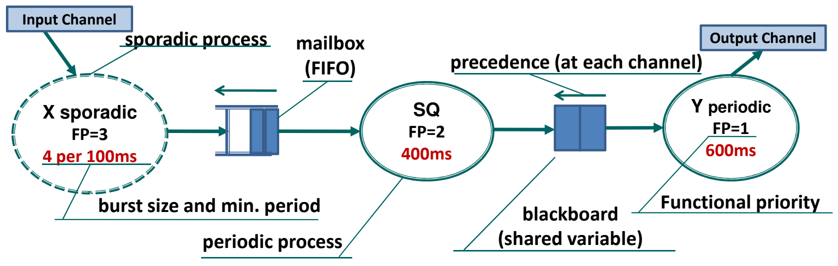

Figure 1: Example Fixed Priority Process Network

The process network in Fig. 1, represents a data processing application, where the "X" sporadic process generates values, the "Square" process calculates the square of the received value and the "Y" periodic process serves as a sink for the squared value. A sporadic event (a command from the environment) activates "X" sporadic, which is annotated by its minimal inter-arrival time. The periodic processes are annotated by their periods. The two types of non-blocking inter-process channels are also illustrated. The FIFO (or mailbox) has a semantics of a queue. The blackboard remembers the last written value that can be read multiple times. The arc depicted above the channels indicates the functional priority relation FP (higher to lower). Additionally, the environment input/output channels are shown. In this example, the dataflow in the channels goes in the opposite direction of the functional priority order.

To design FPPN models, we propose using the architecture description interface of the TASTE toolset [13], whose front-end tools have been amended appropriately. The FPPN model is conceptualized and specified in the TASTE-IV (TASTE Interface-View) graphical editor, which generates an AADL (Architecture Analysis & Design Language) syntax [5] description of the graphical model representation.

In TASTE-IV, the TASTE functions are assigned attributes. The `FPPNClass` attribute defines the type of FPPN entities (e.g. blackboard, periodic process). The sporadic process is configured by coupling two TASTE functions: a function with `FPPNClass = sporadic` that provides a sporadic interface and a function with `FPPNClass=sporadic-protocol` that provides a periodic interface for polling for conditions to invoke the sporadic interface. To satisfy the precondition for the schedulability analysis [16], we assume that one and only one periodic process is connected to each sporadic process, i.e. there is always a single channel that connects a sporadic process to a periodic process. The FPPNClass attributes `mailbox` and `blackboard` are used for data-channels of the respective type; each channel declares two provided interfaces for 'read' and 'write', while the processes that access the channel have respective required interfaces. The `DataChannelSize` should be also defined, which represents the minimum size of the data type (in bytes) communicated via the channel. For a mailbox channel, the `DataChannelLength` is defined, which determines the length of the FIFO.

The `Fpriority` attribute is an integer, which dictates the priority index of the process, and hence its priority order in the network. Each process is assigned a unique index. For any two processes, the one with the larger index has lower functional priority than the other.

In the TASTE-DV (Data View) editor, the designer specifies in ASN.1 language format the types of exchanged data via FPPN channels. This specification is independent from the implementation languages and platforms, and allows deriving automatic marshallers that follow any kind of binary encoding rules.

C code skeletons are generated from the TASTE I-V model together with an XML file that specifies the system topology. The functional code of the application under design employs FPPN templates that reflect in C language the primitives of FPPN processes and channels. The designer can use the TASTE Deployment-View for compiling, binding and running the functional code of the application on his workstation development environment. A first evaluation of the application's functionality is thus possible, without having scheduled its tasks according to their functional priority order and their real-time constraints.

Example 1. *A Guidance Navigation & Control (GNC) on-board spacecraft application controls the movement of the vehicle by processing the data of the corresponding sensors and controllers. GNC involves three steps: the guidance equipment and software first compute the orbital location required to satisfy mission requirements, the navigation then tracks the vehicle's actual location, and the flight control directs the orbit to the required location. Depending on the specific phase, some components can be inactive, and/or the specific data to be exchanged can have a different format. For example, during the orbital phase, the guidance function will provide inertial reference attitude to the controller, whereas in the re-entry phase reference aerodynamic angles will be sent. The application comprises:*

- *The **Guidance Navigation Task** that estimates the current translational state of the vehicle based on measurements by the sensors and on actuator commands, whenever applicable. This task keeps the vehicle on track during the flight to reach the desired location for parachute triggering. It calculates the actual location and provides the reference attitude and the calculated air data and aerodynamic parameters to the control task, which in turn assures its objective. If the reference attitude is pre-computed (e.g. coming from reference trajectory for the Orbital phase), it will also pass through this block to keep the function's definition general. This is a periodic process with period T_p = 500ms, deadline d_P = 500ms and worst-case execution time C=22ms.*
- *The **Control FM task** that runs the control and flight management algorithms. This is a periodic process with period T_p = 50ms, deadline d_P = 50ms and worst-case execution time C=8ms.*
- *The **Control Output Task**, which sends the outputs of the GNC (geodetic altitude, longitude, mach and dynamic pressure) to the Dynamics Kinematics and Environment module. This is a periodic process with period T_p = 50ms, deadline d_P = 50ms and worst-case execution time C=4ms.*
- *The **Data Input Dispatcher Task**, which reads, decodes and dispatches data to the right destination whenever new data from the spacecraft's sensors are available. In our GNC model the data input dispatcher processes MVM (Mission and Vehicle Management), IMU (Inertial Measurement Unit) and GPS (Global Positioning System) data, which have been pre-computed through measurements by their relevant subcomponents and are stored in C buffers. This is a periodic process with period T_p = 50ms, deadline d_P = 50ms and worst-case execution time C=6ms.*

Fig. 2 delineates the exchange of information between GNC tasks in a single hyperperiod of 500ms (least common multiple of all periods). The FPPN model in Fig. 3 was designed based on the Message Sequence Chart and the source code, for identifying how the different tasks interact by conditional signals. The functional priorities of the FPPN impose precedence from numerically smaller (higher-priority) to numerically larger. Fig. 4 depicts the TASTE-IV FPPN model for the GNC application.

$\triangle$

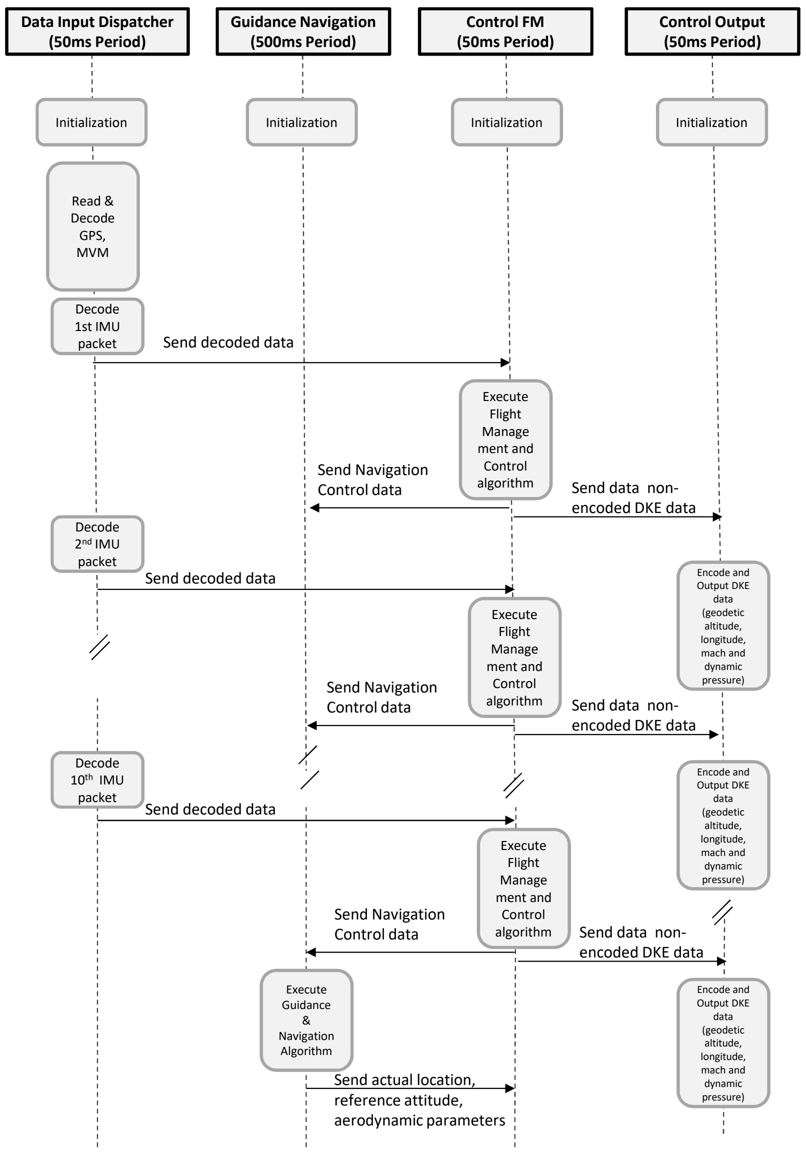
Data Input Dispatcher (50ms Period)
Guidance Navigation (500ms Period)
Control FM (50ms Period)
Control Output (50ms Period)
Initialization
Read & Decode GPS, MVM
Decode 1st IMU packet
Send decoded data
Execute Flight Management and Control algorithm
Send Navigation Control data
Send data non-encoded DKE data
Decode 2nd IMU packet
Encode and Output DKE data (geodetic altitude, longitude, mach and dynamic pressure)
Decode 10th IMU packet
Execute Guidance & Navigation Algorithm
Send actual location, reference attitude, aerodynamic parameters

Figure 2: The Message Sequence Chart of the GNC application

4 Rigorous design of embedded real-time systems based on FPPNs

The proposed design approach aims to a formal and accountable process for deriving implementations of FPPNs whose schedulability is preserved throughout the process and it is eventually guaranteed by construction on top of the real-time BIP execution engine. A series of automated model transformation steps is applied starting with the TASTE2BIP[1] transformation [7] that compiles any FPPN process network into the BIP language. The source code is parsed, searching for primitives (reads and writes from/to the data channels) that are relevant for the process interactions. In addition to the BIP models, the TASTE2BIP model transformation also derives a task graph for static scheduling, as detailed in [16].

[1]TASTE2BIP is online in: `http://www-verimag.imag.fr/Time-Critical-Applications-on-Multicore`

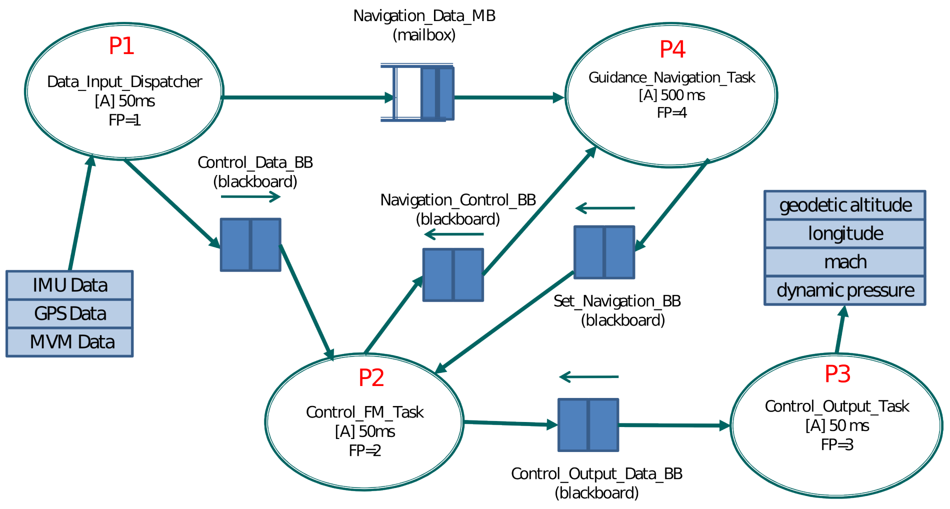

Figure 3: The GNC FPPN model

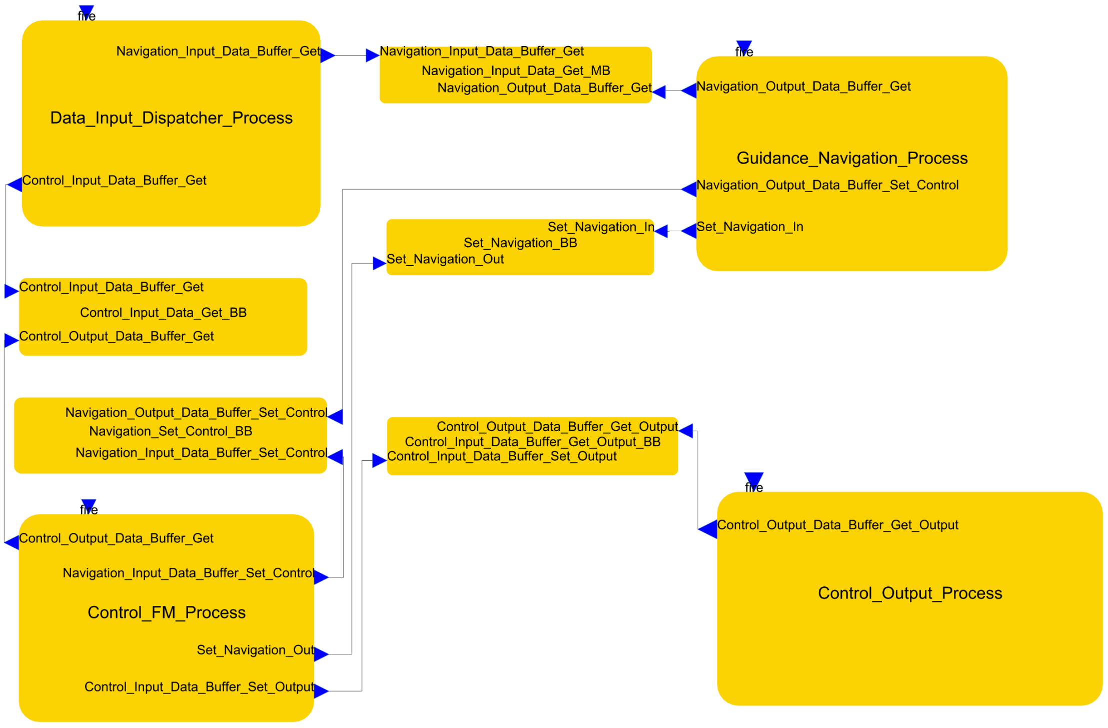
fire
Navigation_Input_Data_Buffer_Get
Data_Input_Dispatcher_Process
Control_Input_Data_Buffer_Get
Navigation_Input_Data_Buffer_Get
Navigation_Input_Data_Get_MB
Navigation_Output_Data_Buffer_Get
fire
Navigation_Output_Data_Buffer_Get
Guidance_Navigation_Process
Navigation_Output_Data_Buffer_Set_Control
Set_Navigation_In
Set_Navigation_In
Set_Navigation_BB
Set_Navigation_Out
Control_Input_Data_Buffer_Get
Control_Input_Data_Get_BB
Control_Output_Data_Buffer_Get
Navigation_Output_Data_Buffer_Set_Control
Navigation_Set_Control_BB
Navigation_Input_Data_Buffer_Set_Control
Control_Output_Data_Buffer_Get_Output
Control_Input_Data_Buffer_Get_Output_BB
Control_Input_Data_Buffer_Set_Output
fire
Control_Output_Data_Buffer_Get
Navigation_Input_Data_Buffer_Set_Control
Control_FM_Process
Set_Navigation_Out
Control_Input_Data_Buffer_Set_Output
fire
Control_Output_Data_Buffer_Get_Output
Control_Output_Process

Figure 4: The TASTE-IV FPPN model for the GNC application

Definition 1 (Task Graph)**.** *A directed acyclic graph* $TG(J,E)$ *whose nodes* $J = \{J_i\}$ *are jobs defined by tuples* $J_i = (p_i, k_i, A_i, D_i, C_i)$*, where* p_i *is the job's process,* k_i *is the job's invocation count,* $A_i \in Q_{\geq 0}$ *is the arrival time,* $D_i \in Q_+$ *is the absolute deadline and* $C_i \in Q_+$ *is the WCET. The k-th job of process* p *is denoted by* $p[k]$*. The edges* E *represent constraints on the job execution order.*

Example 2. *Fig. 5 depicts the task graph derived from the GNC FPPN model of Example 1.* △

Figure 6 introduces the design flow steps and the tool-support associated with each step.

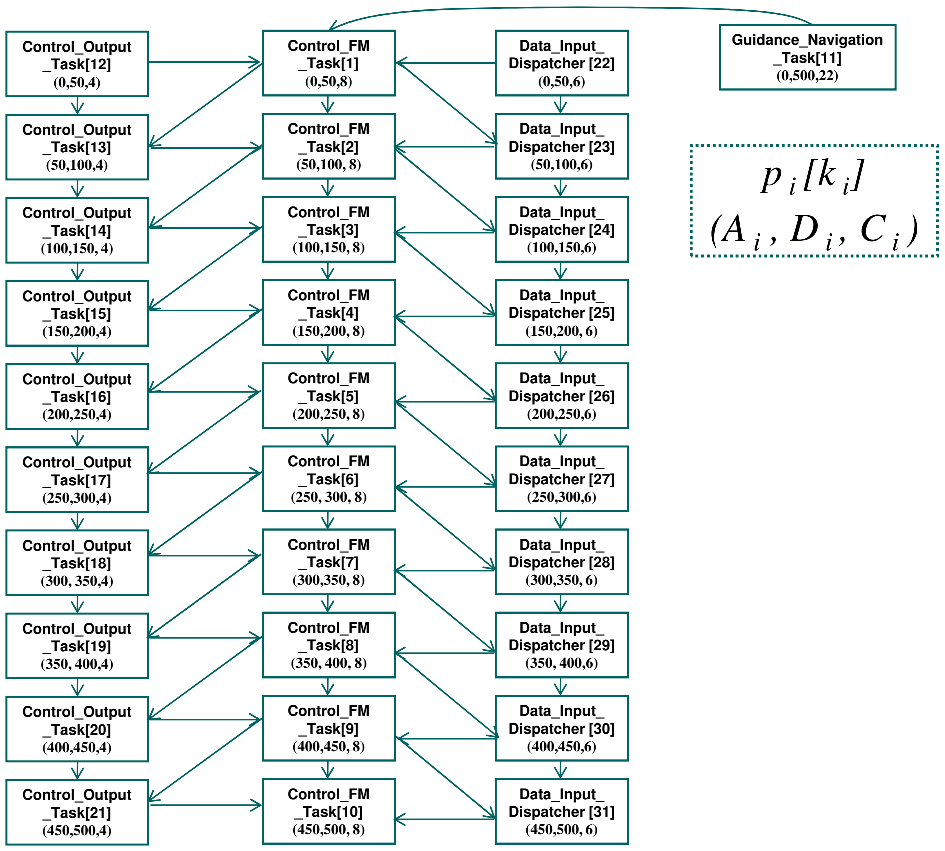
Control_Output_Task[12] (0,50,4)
Control_FM_Task[1] (0,50,8)
Data_Input_Dispatcher [22] (0,50,6)
Guidance_Navigation_Task[11] (0,500,22)
Control_Output_Task[13] (50,100,4)
Control_FM_Task[2] (50,100, 8)
Data_Input_Dispatcher [23] (50,100,6)
$p_i[k_i]$
(A_i, D_i, C_i)
Control_Output_Task[14] (100,150, 4)
Control_FM_Task[3] (100,150, 8)
Data_Input_Dispatcher [24] (100,150,6)
Control_Output_Task[15] (150,200,4)
Control_FM_Task[4] (150,200, 8)
Data_Input_Dispatcher [25] (150,200, 6)
Control_Output_Task[16] (200,250,4)
Control_FM_Task[5] (200,250, 8)
Data_Input_Dispatcher [26] (200,250,6)
Control_Output_Task[17] (250,300,4)
Control_FM_Task[6] (250, 300, 8)
Data_Input_Dispatcher [27] (250,300,6)
Control_Output_Task[18] (300, 350,4)
Control_FM_Task[7] (300,350, 8)
Data_Input_Dispatcher [28] (300,350, 6)
Control_Output_Task[19] (350, 400,4)
Control_FM_Task[8] (350, 400, 8)
Data_Input_Dispatcher [29] (350, 400,6)
Control_Output_Task[20] (400,450,4)
Control_FM_Task[9] (400,450, 8)
Data_Input_Dispatcher [30] (400,450,6)
Control_Output_Task[21] (450,500,4)
Control_FM_Task[10] (450,500, 8)
Data_Input_Dispatcher [31] (450,500, 6)

Figure 5: Task graph for the GNC FPPN model

Input: (i) application requirements (SW items, real-time constraints)
(ii) platform requirements (HW platform & resources)

Output: implementation on the target platform

Step 1 *Architectural design:* The functional code (software behaviour) is implemented and the requirements are mapped to an architectural model (i.e., TASTE I-V). This encompasses the static architecture (static decomposition into software components) and the dynamic architecture (active objects such as threads, tasks and processes, along with their resource and I/O dependencies).

Step 2 *Model transformation:* FPPN model transformation into RT-BIP according to the FPPN execution semantics in [7]. If WCETs are known, the task graph is also generated.
[*if (Task Graph exists) goto Step 5*]

Step 3 *Functional simulation of RT-BIP model:* The application's processor time requirements are judged on the basis of the BIP model's execution on the target platform (Step 4); before this, the same model should be functionally tested on a workstation.

Step 4 *Worst Case Execution Time (WCET) Estimation: The probabilistic measurement-based timing analysis in [15] is used that can arguably guarantee safe probabilistic bounds.*

4.1 *Program instrumentation: A trace point is inserted at every branching of control flow of the task code. The program is run on a workstation to collect execution traces, which are used to identify all potential predictors.*

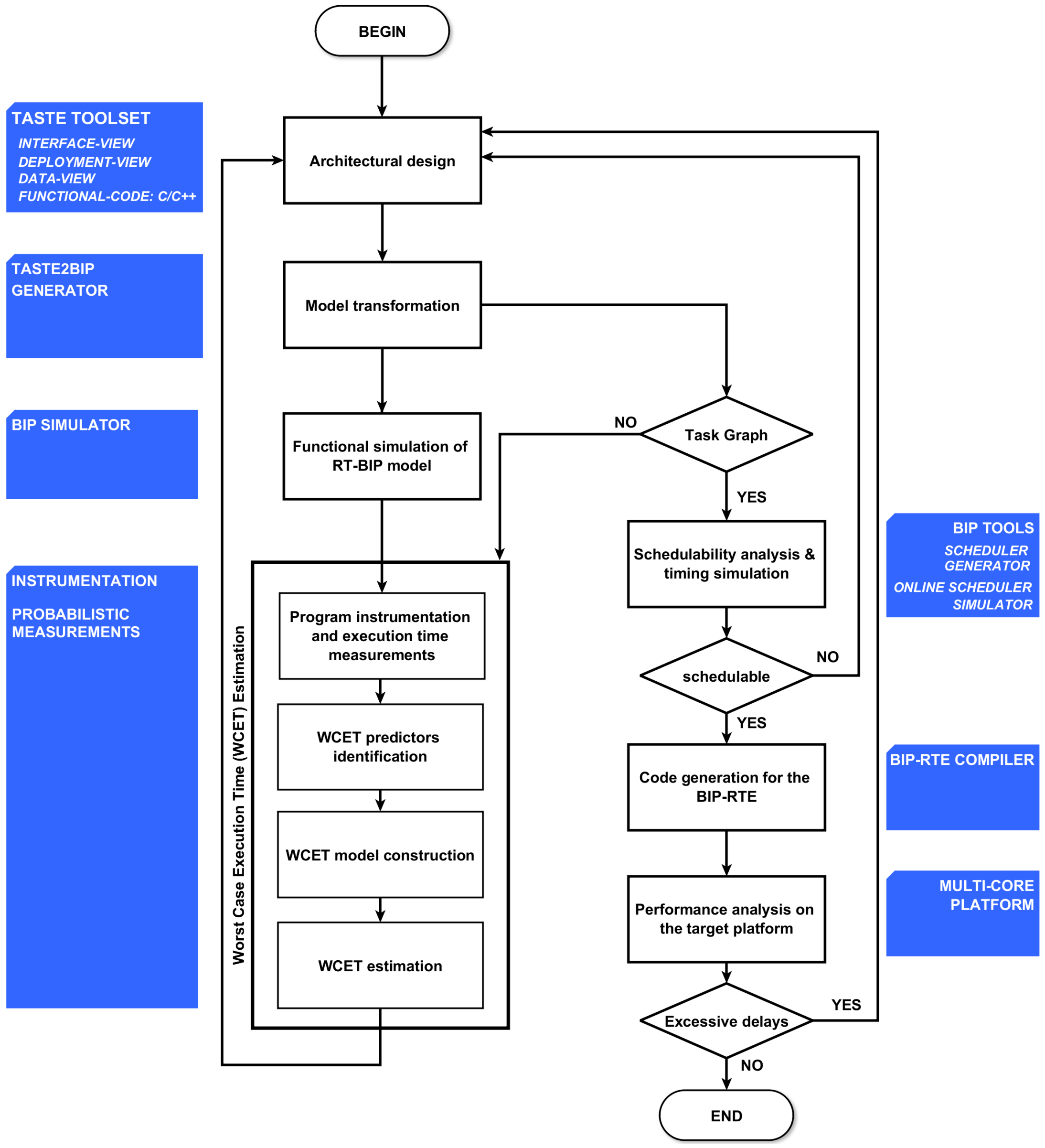
BEGIN
TASTE TOOLSET
INTERFACE-VIEW
DEPLOYMENT-VIEW
DATA-VIEW
FUNCTIONAL-CODE: C/C++
Architectural design
TASTE2BIP GENERATOR
Model transformation
BIP SIMULATOR
Functional simulation of RT-BIP model
NO
Task Graph
YES
Schedulability analysis & timing simulation
BIP TOOLS
SCHEDULER GENERATOR
ONLINE SCHEDULER SIMULATOR
INSTRUMENTATION
PROBABILISTIC MEASUREMENTS
Worst Case Execution Time (WCET) Estimation
Program instrumentation and execution time measurements
WCET predictors identification
WCET model construction
WCET estimation
schedulable
NO
YES
Code generation for the BIP-RTE
BIP-RTE COMPILER
Performance analysis on the target platform
MULTI-CORE PLATFORM
Excessive delays
YES
NO
END

Figure 6: Rigorous design flow for embedded systems based on the real-time BIP execution engine

4.2 *WCET predictors identification: A sufficient subset of the set of potential predictors is identified for an adequate execution time regression model.*

4.3 *WCET model construction: The Maximal Regression Model for conservative overestimation of the execution time is applied [15].*

4.4 *WCET estimation: A Maximal Execution Time bound is computed.*
[*goto Step 1*]

Step 5 *Schedulability analysis & timing simulation:* The task graph is given as input to a static scheduler, along with the attributes of each job (the process to which the job belongs, its invocation count, arrival time, absolute deadline, WCET).

[*if (! schedulable) iterate Steps 1 to 4*]

Step 6 *Code generation for the BIP RTE:* The joint application/scheduler model is compiled by the RT BIP compiler and linked with the BIP-RTE.

Step 7 *Performance analysis on the target platform:* Validation by performance analysis is essential towards identifying possible excessive delays, due to resource starvation cases.

[*if (excessive delays found) goto Step 1*]

Fig. 6 shows the tools used in each step. Apart from the TASTE2BIP transformation, functional simulation in step 3 takes place using the RT-BIP tools. The statistical tools for the WCET estimation in step 4 are described in [15]. In step 5, the schedule obtained from the scheduler is translated into input for the online-scheduler model in BIP, which constraints job executions for resource management (task to processor core mapping and other constraints). In steps 4.3 and 7, the executable runs on the target platform on top of the real-time operating system (RTEMS-SMP). For the analysis in step 7, appropriate tools are used (e.g. gprof) that trace/monitor the software performance on the target platform.

5 Schedulability analysis and code generation for the BIP RTE

To illustrate the main principles of FPPN scheduling we consider the synthetic application in Fig. 7 with three tasks. The “split” task appends two small data items (a few bytes) to the two output channels and sleeps for 1 ms to imitate some task execution time. Tasks “A” and “B” read the data and Task A sleeps for 12 ms whereas Task “B” sleeps for 6 ms. All tasks have the same periodic scheduling window, with period and deadline being 25 ms. In a real application, this corresponds to the time during which the two input data buffers should be read, the computations performed and the output buffers written. In the derived task graph, every task is represented by a job. The jobs are numbered as $J_i = J_1, J_2, J_3$ and annotated by their WCETs. The arrival times A_i and deadlines D_i for all jobs are the same.

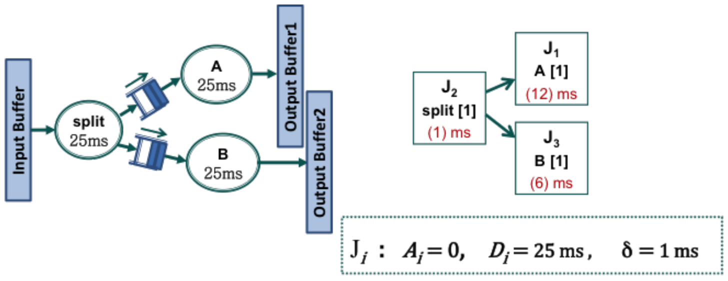

Figure 7: The FPPN model of a system with three tasks

In step 5 of the design flow, the static scheduler accepts a parameter δ for the worst-case cost of a single transition in the BIP automata components. Parameter δ is platform-dependent and characterizes the coarse-grain interference between the task components, when they access the centralized BIP RTE engine to execute discrete automata transitions. In the simplest possible scheduling, every job is executed as soon as it arrives and its predecessors have finished (ASAP policy). In this case, the functional priority ordering (`Fpriority` attributes in the TASTE-IV FPPN) that is implemented in the generated RT-BIP model enforces the predecessor - successor relation between the jobs thus ensuring deterministic updates to the channel states.

The scheduler tool produces the time-triggered table depicted in Fig. 8 with the times when discrete BIP transitions occur, which are imposed by the online scheduler. The blocks executed on Core 0

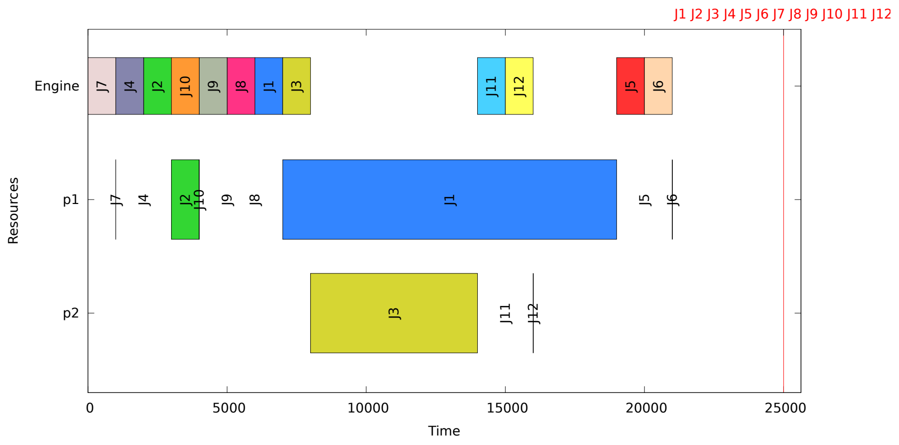

Figure 8: FPPN schedule computed by the static scheduler for the three-tasks example

correspond to discrete BIP transitions of duration $\delta = 1ms$.

The tool applies list scheduling based on an heuristically computed priority relation (consistent with the functional priority relation), a total order in which earlier jobs have higher priority. This involves a simple simulation of the fixed-priority policy [16]. For the task graph of Fig. 7 two compute cores are needed. This happens due to the 12 ms interference overhead (four BIP transitions required per task's execution, take $4{\cdot}\delta = 4$ ms); the task graph cannot be scheduled on a single processor, because an amount 12+(1+12+6) ms = 31 ms of processor time per period of 25 ms has to be allocated. In the schedule of Fig. 8, Task split and Task A are mapped to Core 1 and Task B to Core 2.

An expressive timed automata language like BIP provides potential means to implement custom strategies for controlling interference on shared HW/SW resources. As we have seen in the above example, interference has to be taken into account together with the WCET in the schedulability model. It can be controlled through adoption of an appropriate interference model, as shown by the authors of [14]. In that work, additional schedulability perspectives are also considered, most notably mixed-criticality scheduling.

6 The GNC application with BIP RTE running on a quad-core platform

Through the rigorous design flow we had convenient means to experiment with various scheduling scenarios when running the GNC application with BIP RTE on a LEON4FT embedded platform. Our aim was to explore the speedup in throughput potential for each scenario. We present here the results for a "pipelined" scenario which represented an attempt to exploit multi-core parallelism as much as possible; the potential for such an exploratory approach is inherent in the FPPN model and remains transparent to the application designer until the final steps of our rigorous design flow.

Specifically, we focused on the functional priority relation of Fig. 3, where the Control Output (P3) and Guidance Navigation (P4) tasks process the data received in the previous period. Therefore, they both have enough data to start execution immediately at the beginning of the period. Thus, the priority relation, which should normally for this application follow in Fig. 3 the direction from left to right, for the channels connected to these two tasks, follows the opposite direction. Note that the mailbox channel even has no functional priority arrow associated with it, which will be justified below. Thus, P3 and P4 have no predecessors, and the same holds for P1, and the three of them can run in parallel. The absence of arrow for the mailbox between P1 and P4, which makes it possible for them to run in parallel, is explained as follows. Normally the data channels should be protected by priority arrow to avoid data races, but instead we increased the capacity of the mailbox to make a double buffer out of it. Thus, the reader process, P4, can read one data item, whereas the writer process, P1, writes the other one, and they can do it concurrently without risking data races (but with risking some interference, as we see later on).

Fig. 9 shows the Gantt chart, which zooms in the time-window, where the jobs are executed in parallel on the 3 cores plus the fourth core (core 0) where the BIP RTE engine executes the BIP transitions.

We see that all activities in the current period end by time slightly less than 40 ms after the period start. This gives us a (reverse proportional) measure of a maximal throughput that the system could achieve, by reducing the period from 50 ms to 40 ms. Despite a higher parallelism compared to a scenario where three cores are utilized [7], no improvement is achieved in throughput due to the interference between P1, P4, which manifested in longer execution time of P4. A possible solution to this end is modifying the mailbox BIP implementation to reduce the interference.

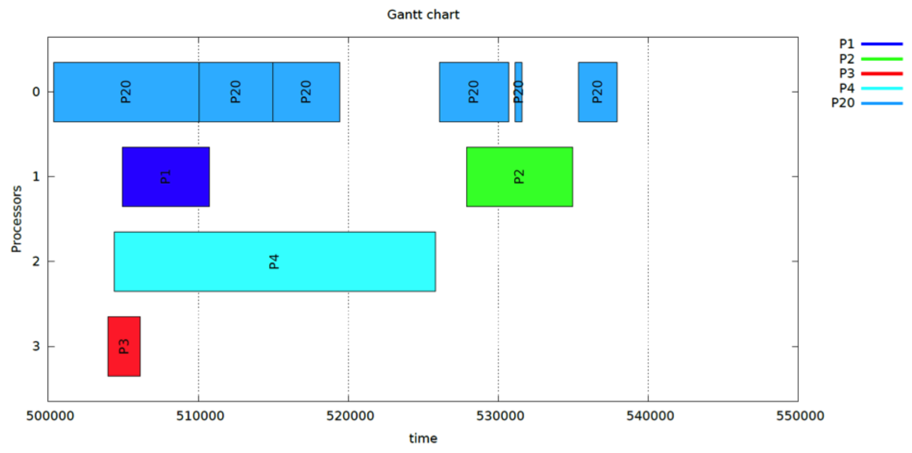

Figure 9: Zoom in a time window with parallel job executions of the GNC on the LEON4FT processor

7 Conclusion

We introduced a rigorous approach for multi-core embedded system design based on the FPPN process network model and the BIP RTE. FPPNs allow reasoning in terms of high-level schedulability concepts, whereas predictability on the execution platform is guaranteed by construction, since the BIP RTE enables the consistent mapping of user-programmed scheduling policies to operating system mechanisms (e.g. threads, dynamic priorities).

References

[1] *GR-CPCI-LEON4-N2X: Quad-Core LEON4 Next Generation Microprocessor Evaluation Board, http://www.gaisler.com/index.php/products/boards/gr-cpci-leon4-n2x.*

[2] T. Abdellatif, J. Combaz & J. Sifakis (2010): *Model-based implementation of real-time applications*. In: *EMSOFT '10*.

[3] G. Brau, J. Hugues & N. Navet (2018): *Towards the systematic analysis of non-functional properties in Model-Based Engineering for real-time embedded systems*. Science of Computer Programming 156, pp.

1 – 20, doi:https://doi.org/10.1016/j.scico.2017.12.007. Available at `http://www.sciencedirect.com/science/article/pii/S0167642317302927`.

[4] J. Eker, J. W. Janneck, E. A. Lee, Jie Liu, Xiaojun Liu, J. Ludvig, S. Neuendorffer, S. Sachs & Yuhong Xiong (2003): *Taming heterogeneity - the Ptolemy approach*. *Proceedings of the IEEE* 91(1), pp. 127–144, doi:10.1109/JPROC.2002.805829.

[5] P. Feiler, D. Gluch & J. Hudak (2006): *The Architecture Analysis & Design Language (AADL): An Introduction*. Technical Report CMU/SEI-2006-TN-011, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. Available at `http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=7879`.

[6] F. Gioulekas, P. Poplavko, R. Kahil, P. Katsaros, M. Bozga, S. Bensalem & P. Palomo (2017): *Design of Embedded Systems with Complex Task Dependencies and Shared Resource Interference (Short Paper)*. In: *Software Engineering and Formal Methods - 15th International Conference, SEFM 2017, Trento, Italy, September 4-8, 2017, Proceedings*, pp. 401–407, doi:10.1007/978-3-319-66197-1_28. Available at `https://doi.org/10.1007/978-3-319-66197-1_28`.

[7] F. Gioulekas, P. Poplavko, P. Katsaros, S. Bensalem & P. Palomo (2018): *A Process Network Model for Reactive Streaming Software with Deterministic Task Parallelism*. In: *Fundamental Approaches to Software Engineering (FASE)*, *LNCS* , Springer.

[8] S. Ha, S. Kim, C. Lee, Y. Yi, S. Kwon & Y. Joo (2008): *PeaCE: A Hardware-software Codesign Environment for Multimedia Embedded Systems*. *ACM Trans. Des. Autom. Electron. Syst.* 12(3), pp. 24:1–24:25, doi:10.1145/1255456.1255461. Available at `http://doi.acm.org/10.1145/1255456.1255461`.

[9] N. Halbwachs (2010): *Synchronous Programming of Reactive Systems*. Springer-Verlag, Berlin, Heidelberg.

[10] N. Halbwachs & P. Raymond (1999): *Validation of Synchronous Reactive Systems: From Formal Verification to Automatic Testing*. In: *Advances in Computing Science - ASIAN'99, 5th Asian Computing Science Conference, Phuket, Thailand, December 10-12, 1999, Proceedings*, pp. 1–12, doi:10.1007/3-540-46674-6_-1. Available at `https://doi.org/10.1007/3-540-46674-6_1`.

[11] J. Hugues, B. Zalila, L. Pautet & F. Kordon (2008): *From the Prototype to the Final Embedded System Using the Ocarina AADL Tool Suite*. *ACM Trans. Embed. Comput. Syst.* 7(4), pp. 42:1–42:25, doi:10.1145/1376804.1376810. Available at `http://doi.acm.org/10.1145/1376804.1376810`.

[12] A. Kanduri, A. M. Rahmani, P. Liljeberg, K. Wan, K. L. Man & J. Plosila (2013): *A multicore approach to model-based analysis and design of Cyber-Physical Systems*. In: *2013 International SoC Design Conference (ISOCC)*, pp. 278–281, doi:10.1109/ISOCC.2013.6864027.

[13] M. Perrotin, E. Conquet, J. Delange, A. Schiele & T. Tsiodras (2012): *TASTE: A Real-Time Software Engineering Tool-Chain Overview, Status, and Future*. In I. Ober & I. Ober, editors: *SDL 2011: Integrating System and Software Modeling: 15th International SDL Forum Toulouse, France, July 5-7, 2011. Revised Papers*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 26–37.

[14] P. Poplavko, R. Kahil, D. Socci, S. Bensalem & M. Bozga (2016): *Mixed-Critical Systems Design with Coarse-Grained Multi-core Interference*. In: *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques - 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10-14, 2016, Proceedings, Part I*, pp. 605–621.

[15] P. Poplavko, A. Nouri, L. Angelis, A. Zerzelidis, S. Bensalem & P. Katsaros (2017): *Regression-Based Statistical Bounds on Software Execution Time*. In: *Verification and Evaluation of Computer and Communication Systems - 11th International Conference, VECoS 2017, Montreal, QC, Canada, August 24-25, 2017, Proceedings*, pp. 48–63. Available at `https://doi.org/10.1007/978-3-319-66176-6_4`.

[16] P. Poplavko, D. Socci, P. Bourgos, S. Bensalem & M. Bozga (2015): *Models for deterministic execution of real-time multiprocessor applications*. In: *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1665–1670.

[17] I. Radojevic & Z. Salcic (2011): *Embedded Systems Design Based on Formal Models of Computation*, 1st edition. Springer Publishing Company, Incorporated.

[18] K. Schneider, J. Brandt & E. Vecchie (2006): *Efficient code generation from synchronous programs*. In: *Fourth ACM and IEEE International Conference on Formal Methods and Models for Co-Design, 2006. MEMOCODE '06. Proceedings.*, pp. 165–174, doi:10.1109/MEMCOD.2006.1695922.

[19] J. Sifakis (2013): *Rigorous System Design*. *Foundations and Trends in Electronic Design Automation* 6(4), pp. 293–362, doi:10.1561/1000000034. Available at `http://dx.doi.org/10.1561/1000000034`.