Design of embedded systems with complex task dependencies and shared resource interference (Short Paper)*

F. Gioulekas[1,5] and P. Poplavko[2], R. Kahil[2], P. Katsaros[1,4], M. Bozga[2], S. Bensalem[2] and P. Palomo[3]

[1] Information Technology Institute, CERTH, Greece
`gioulekas@teemail.gr`
[2] Universite Joseph Fourier - Verimag, Grenoble, France
`petro.poplavko, rany.kahil, marius.bozga, saddek.bensalem@imag.fr`
[3] Deimos-Space S.L.U, Madrid, Spain
`pedro.palomo@deimos-space.com`
[4] Aristotle University of Thessaloniki, Greece
`katsaros@csd.auth.gr`
[5] Current affiliation: University General Hospital of Larissa, Greece

Abstract. Languages for embedded systems ensure predictable timing behavior by specifying constraints based on either data streaming or reactive control models of computation. Moreover, various toolsets facilitate the incremental integration of application functionalities and the system design by evolutionary refinement and model-based code generation. Modern embedded systems involve various sources of interference in shared resources (e.g. multicores) and advanced real-time constraints, such as mixed-criticality levels. A sufficiently expressive modeling approach for complex dependency patterns between real-time tasks is needed along with a formal analysis of models for runtime resource managers with timing constraints. Our approach utilizes a model of computation, called Fixed-Priority Process Networks, which ensures functional determinism by unifying streaming and reactive control within a timed automata framework. The tool flow extends the open source TASTE tool-suite with model transformations to the BIP language and code generation tools. We outline the use of our flow on the design of a spacecraft on-board application running on a quad-core LEON4FT processor.

Keywords: model-based design, embedded systems, model of computation, code generation, multicores.

1 Introduction

The model-based design philosophy for embedded systems is grounded on the evolutionary design using models [4], which support the analysis, the gradual

* This work was supported by ESA under contract No. 4000111814/14/NL/MH. This is a short paper accepted in the new ideas and work-in-progress section of SEFM 2017.

refinement and the setting of real-time attributes that ensure predictable timing behavior. For being able to analyze the models, they are specified with languages based on formal models of computation [1], which allow the synthesis and the optimization of behavior into an implementation solution. Such models provide syntax for describing dependencies between the runtime entities of a design and rules for computation of the behavior, given the syntax. The well-known streaming models of computation are suitable for describing complicated data transfer functions, whereas the reactive control models used in synchronous languages are suitable for complex control dependencies, which are compiled to sequential code as tasks, and classical schedulability methods can then be applied.

However, in modern embedded systems the task dependencies are further complicated, due to various sources of interference in shared software and hardware resources (e.g. buses, DMAs, I/Os in multicores) and additional constraints, such as mixed-criticality levels, dynamic voltage and frequency scaling. Thus, the design should ensure predictable timing behavior, while allowing adaptation to unexpected overload cases by dynamically reallocating resources.

To this end, we present a rigorous design approach that integrates a recently introduced model of computation, the Fixed Priority Process Networks (FPPNs) [3], with the TASTE toolset [5] and a timed automata analysis framework with "resource managers" [6], i.e. software functions that monitor utilization of compute resources and adapt the schedule in cases of shortage [2].The FPPN model of computation combines the expressiveness of streaming and reactive control, retains the efficiency potential of parallel processing and ensures functional determinism, i.e. the program's outputs are neither dependent on the tasks' execution times, nor on their scheduling. TASTE is an open source toolset based on a system-level architecture description language, the AADL. It supports the incremental model-based integration (through ASN.1) of application functionalities using various languages (C/C++, SDL, VHDL, Ada, Python) and tools (SCADA, Simulink). We utilized TASTE's extensibility support towards enabling the design of FPPN programs. Moreover, a model transformation was implemented to a timed automata modeling framework in BIP, a language with formal operational semantics and code generation tools, for execution engines ported to various embedded platforms. Our approach allows scheduling the program's tasks, while taking into account their dependencies and the various sources of interference, through explicit interference models and resource managers. A resource manager is an integral part of an online scheduler that implements a customized online scheduling policy.

We present the scheduling of a Guidance, Navigation and Control (GNC) application on the quad-core LEON4FT in ESA's Next Generation Microprocessor platform (NGMP) [7]. Section 2 summarizes background knowledge on the FPPN model of computation. Section 3 introduces the TASTE toolset extensions to support FPPNs, the TASTE2BIP model transformations and task graph extraction, which enable the application's scheduling based on appropriate interference models. Section 4 presents the scheduling of the GNC application and the paper concludes with an overview of the exposed contributions.

2 Fixed Priority Process Networks

The FPPN [3] extends the reactive control models of computation by introducing synchronization and pipelined execution for a set of processes (tasks), which communicate data through channels. It allows the specification of time dependent, yet deterministic, behavior and real time task properties (sporadic or periodic activations with deadlines), and can be scheduled on single or multiple processors with or without priorities. The determinism is ensured by a *functional priority* relation between the tasks that are executed in an order, which is determined first by the task release times, i.e. when the tasks are invoked, and secondly by the task priorities.

An FPPN consists of *processes*, *data channels* and *event generators*. The processes represent subroutines with *functional code* featuring internal variables and ports connected to their input/output channels. A subroutine invocation is defined as a job with bounded execution time, which is subject to worst-case execution time (WCET) analysis. Every process is associated with an event generator, which can be either periodic or sporadic. The data channels support non-blocking read and write operations, which means that reading from an empty channel does not block the reader. The returned data value is accompanied by a validity flag, i.e. a boolean indicator of whether the data is valid. There are inter-process and external (environment) channels of two possible types, FIFO or blackboard. The blackboard remembers the last written value, which can be read multiple times.

Every process p has a deadline d_p. An event generator's sequence of timestamps τ_k determines when the kth job of process p is "activated". The periodic processes are activated with period T_p, whereas for sporadic processes T_p denotes the minimum inter-arrival time. Each job's execution has to be completed by $D_k = \tau_k + d_p$. We assume that all simultaneous process activations are signaled synchronously and we consider two variants of FPPN semantics. According to the *zero-delay* semantics the processes' execution takes zero time and since all deadlines can be met without exploiting parallelism, we assume for simplicity that it takes place sequentially. The deterministic ordering of non-blocking accesses to the shared variables between the processes is ensured by a set of rules detailed in [3]. The zero-delay semantics allows the functional simulation of the FPPN through its sequential execution. The real time semantics defines how the FPPN is executed on embedded platforms, which is a relaxed version of the zero-delay semantics, since it allows jobs to have any execution time, as well as to start concurrently at any time after their invocation.

For certain subclasses of FPPNs it is possible to statically derive a *task graph*, which then serves as input to a scheduling algorithm. A task graph is a directed acyclic graph $TG(J, \mathcal{E})$ with nodes representing jobs $J = \{J_i\}$ and edges $\mathcal{E}$ that are called *precedence edges*, which constrain the job execution order. A job is characterized by a tuple $J_i = (p_i, k_i, A_i, D_i, C_i)$ wherep_i is the process to which the job belongs, k_i is the job's invocation count, $A_i \in Q_{\geq 0}$ is the arrival time, $D_i \in Q_+$ is the required time (absolute deadline) and $C_i \in Q_+$ is the WCET.

3 Design and scheduling for FPPNs in TASTE

Fig. 1 delineates our model based design flow that integrates FPPNs within TASTE [5], along with a timed automata modeling framework in BIP [3] (parts in grey color depict our contribution) and its associated scheduling and code generation tools. The latter ensure predictable timing behavior, when executing the application on a multithreaded BIP Runtime Environment (BIB RTE). Specifically, a representation of the software is provided through the TASTE front-end tools (Interface View, Data View, Deployment View), which were amended to capture FPPN-compliant models. The TASTE functions can be assigned attributes that characterize the FPPN node (e.g. blackboard, periodic process). Each process is associated with a unique integer (larger numbers imply lower priorities) and a criticality level (only HI and LO are supported), for scheduling the application with multiple criticality levels [6]. Additionally, functional C/C++ code primitives are inserted in TASTE including also ASN.1 based data types.

The next step is the TASTE2BIP model transformation, where (i) the TASTE FPPN is transformed to a BIP FPPN model and (ii) the TASTE attributes are used to generate the task graph through graph rewriting. At this stage, we take into account the interference on shared software and hardware resources, which invalidates the canonical WCET and schedulability analysis, due to a feedback influence. This step involves the design of an interference model, as detailed in [6]. The schedule obtained from the static scheduler together with the interference model are then translated into parameters of the online-scheduler model in BIP. The joint application and scheduler representation is compiled into an executable, which is linked with the resource manager BIP RTE and executed on the target platform on top of the real time operating system.

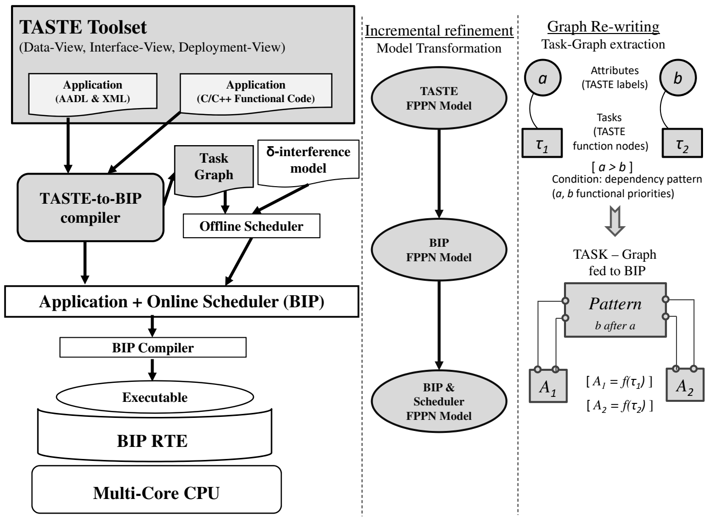

Fig. 1. Model-based design and tool flow for FPPNs in TASTE.

4 Case-Study: Guidance Navigation Control Application

The described approach was applied on the design and scheduling of a GNC application ported onto ESA's NGMP with the aim to utilize multiple cores of the quad-core LEON4FT processor [7]. The main objective of a GNC application is to affect the movement of the vehicle and provide the corresponding sensor and controller with the necessary data. It comprises the Guidance Navigation Task (Functional Priority = 4, Period = 500ms, Deadline = 500ms, WCET=22ms), the Control Output Task (Functional Priority = 3, Period = 50ms, Deadline = 50ms, WCET = 3ms) that sends the outputs to the appropriate spacecraft unit, the Control FM Task (Functional Priority=2, Period=50ms, Deadline=50ms, WCET=8ms) which runs the control and flight management algorithms, and the Data Input Dispatcher Task (Functional Priority=1, Period=50ms, Deadline=50ms, WCET=6ms), which reads, decodes and dispatches data to the right destination whenever new data from the spacecraft's sensors are available.

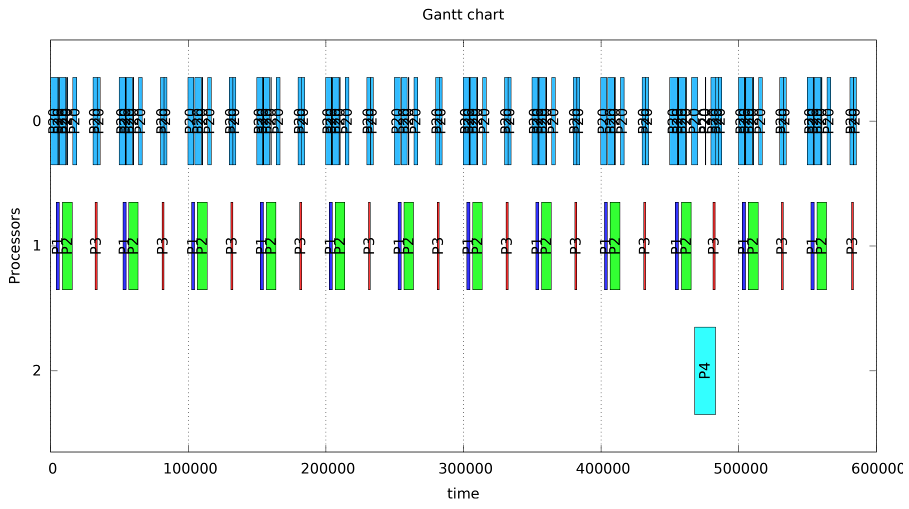

Fig. 2. Execution of the GNC application on LEON4FT (in microseconds).

The TASTE2BIP tool transformed the TASTE Interface View models (XML and C language) to an equivalent FPPN BIP model. The calculated hyper-period (least common multiple of periods) was H = 500ms. The Guidance Navigation and Control Output Tasks start with time offsets 450ms and 30ms, respectively. This information was inserted into the BIP model by manually modifying the default design flow script. The task graph data was then passed to the BIP offline scheduler tool, which estimated the load (utilization) to be 112% (thus requiring two compute cores) and provided the time-triggered scheduling tables. This computation took into account the interference of the BIP engine and the precedence constraints. The last step was to compile the BIP model, to link it with the BIP RTE and to execute it on the quad-core LEON4FT processor. The executables were subsequently loaded and executed on the LEON4FT board. Fig.

2 depicts the execution of the GNC model on the NGMP, within a time frame equal to the hyper-period of 500ms plus another 50ms. The GNC application utilizes one core for the resource manager P20 (BIP RTE and BIP controllers) and two computing cores for the application's tasks. Process P1 corresponds to the Data Input Dispatcher Task, P2 to the Control FM Task, P3 to the Control Output Task and P4 to the Guidance Navigation Task. Minor time shifts to the jobs execution time are noticed and this is due to the P20 overhead. However, runtime overhead is present in every execution environment.

5 Conclusion

A rigorous model-based design flow was introduced for embedded systems with complex task dependencies and shared resource interference, which is integrated with the TASTE toolset. Task dependencies and shared resource interference are arbitrated through dependency patterns according to the FPPN model of computation. Experimental results demonstrated the efficacy of the proposed design flow through the modeling and execution of a GNC application on the quad-core LEON4FT processor. As future work, we intend to support more than two criticality levels, and at the TASTE2BIP model transformation level the use of additional languages such as ITU-T SDL and Simulink. Furthermore, we also intend to utilize more TASTE design capabilities by implementing test-bench wrappers using python test-benching.

References

1. Radojevic, I., Salcic, Z.: Models of Computation and Languages. In: Embedded Systems Design Based on Formal Models of Computation, pp 7 - 41, Springer (2011).
2. Fersman, E., Krcal, P., Pettersson, P., Yi, W.: Task automata: Schedulability, decidability and undecidability. Inf. Comput. 205(8), pp. 1149 1172. Elsevier (2007).
3. Poplavko, P., Socci, D., Bourgos, P., Bensalem, S., Bozga, M.: Models for Deterministic Execution of Real-time Multiprocessor Applications. In: Design, Automation and Test in Europe Conference and Exhibition, pp. 1665 1670, DATE 2015. Grenoble, France (2015).
4. Hugues, J., Zalila, B., Pautet, L., Kordon, F.: From the Prototype to the Final Embedded System Using the Ocarina AADL Tool Suite. ACM Trans. on Emb. Comp. Syst. vol 7(4), pp. 42:1 – 42:25 (2008).
5. Perrotin, M., Conquet, E., Delange, J., Schiele, A., Tsiodras, T.: TASTE: A Real-time Software Engineering Tool-chain Overview, Status, and Future. In: 15th International Conference on System Design Languages, SDL Forum, pp. 26 37. Toulouse, France (2011) .
6. Poplavko, P., Kahil, R., Socci, D., Bensalem, S., Bozga, M.: Mixed-Critical Systems Design with Coarse-Grained Multi-core Interference. In: Margaria T., Steffen B. (eds) Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques, ISoLA 2016. LNCS, vol. 9952. Springer, Cham (2016).
7. GR-CPCI-LEON4-N2X: Quad-Core LEON4 Next Generation Microprocessor Evaluation Board, http://www.gaisler.com/index.php/products/boards/gr-cpci-leon4-n2x.