A Novel Model for System-Level Decision Making with Combined ASP and SMT Solving

Alexander Biewer, Jens Gladigau
Robert Bosch GmbH, Germany
Corporate Sector Research
{alexander.biewer, jens.gladigau}@de.bosch.com

Christian Haubelt
University of Rostock, Germany
Applied Microelectronics and Computer Engineering
christian.haubelt@uni-rostock.de

***Abstract*—In this paper, we present a novel model enabling system-level decision making for time-triggered many-core architectures in automotive systems. The proposed application model includes shared data entities that need to be bound to memories during decision making. As a key enabler to our approach, we explicitly separate computation and shared memory communication over a network-on-chip (NoC). To deal with contention on a NoC, we model the necessary basis to implement a time-triggered schedule that guarantees freedom of interference. We compute fundamental design decisions, namely (a) spatial binding, (b) multi-hop routing, and (c) time-triggered scheduling, by a novel coupling of answer set programming (ASP) with satisfiability modulo theories (SMT) solvers. First results of an automotive case study demonstrate the applicability of our method for complex real-world applications.**

I. Introduction

The increase in system complexity and computational demand of embedded systems is tremendous. In future automotive applications, for example, sophisticated control algorithms are necessary to satisfy mandatory emission targets and feature requests of customers. Though, increasing demands require hardware platforms with more computational performance. Many-core systems with a network-on-chip (NoC) interconnecting the growing number of processing elements are deemed to offer scalable performance [1]. A necessary step in electronic system-level synthesis for such hardware architectures is the process of decision making: fundamental decisions concerning spatial binding of software components, multi-hop routing of communication, and scheduling shared resources have to be made [2]. In the presence of hard real-time constraints, manual decision making approaches are likely to fail or may result in inferior implementations. The reasons are twofold: (1) exploiting massively parallel architectures requires a concurrent application design and the potential for spatial bindings of software components at a fine granular level; as a result, the number of required decisions increases drastically. (2) Decisions are interdependent and affect each other. Consequently, efficient automated decision making has to deal with growing system complexity and intricate relations.

In this paper, we present a novel model with semantics that enable an automated decision making approach for many-core systems with NoC. In contrast to related work [3]–[7], we explicitly include shared data entities in our application model. In the process of decision making, data entities have to be bound to memories. Computational tasks then process the data. With limited local memory, non-local access to shared data via shared hardware resources in the NoC has to be addressed.

As a key enabler to our approach, we present execution semantics that explicitly separate shared memory communication and computation, resulting in an input-processing-output model. Introducing the semantics provides two major benefits: (1) we can integrate state-of-the-art worst-case execution time (WCET) analyzers, e.g. [8], to compute tight WCETs. (2) In combination with a time-triggered schedule, we ensure contention-free access to shared resources.

On basis of the presented model, we propose a novel automated decision making approach by coupling two constraint satisfaction problem (CSP) solvers. Based on the system's specification, a spatial binding of software components, namely computational tasks and data entities, and multi-hop routes of communication transactions are computed using answer set programming (ASP) [9]. With its features to express reachability directly in the modelling language, ASP enables scalable multi-hop routing [3]. On basis of the ASP decisions, a satisfiability modulo theories (SMT) [10] solver calculates a time-triggered schedule satisfying all timing constraints if possible.

Contributions of the paper at hand are as follows:

- We present a novel, formal model for control applications that explicitly separates shared memory communication and computation.
- We explicitly respect the placement of data entities in our model and in our proposed decision making approach.
- We present a novel synthesis approach using a combination of ASP and SMT solving.

II. System Model

The following section introduces the system model which is composed of a platform model and an application model.

A. Platform Model

The homogeneous many-core hardware platform consists of multiple resources of the type *tile*, *router* and *interconnection*. A *tile* integrates one processing element (PE) and a fixed amount S of local memory. Each tile's memory is part of a distributed shared memory that can be addressed directly. Formally, the platform is modeled as a graph $g_P = (\mathbf{R}, \mathbf{E}_p)$. A node $r \in \mathbf{R}$ represents a resource. With focus on tiled architectures, a resource may either be a *tile* $r_t \in \mathbf{R}_t \subseteq \mathbf{R}$ or a *router* $r_r \in \mathbf{R}_r \subseteq \mathbf{R}$. A directed edge $e \in \mathbf{E}_p \subseteq (\mathbf{R} \times \mathbf{R})$ models an unidirectional communication connection between two resources. As an example, Fig. 1 illustrates a hardware

978-3-9815370-2-4/DATE14/© 2014 EDAA

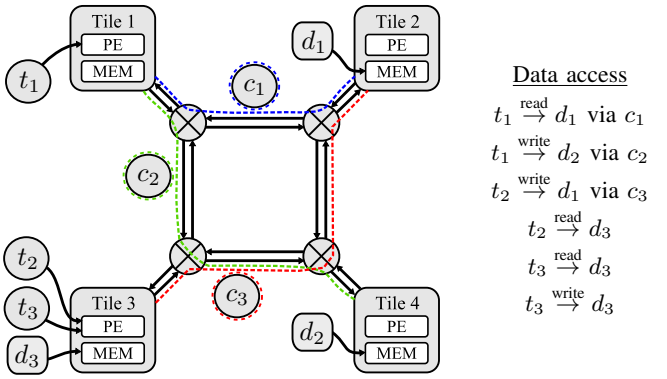

Fig. 1. Motivating example for system-level decision making in the context of the presented system model. Decisions referring to spatial alignment are illustrated: computational tasks t_i are bound to PEs, data entities d_i are bound to memories, and communication task c_i are routed on the network-on-chip (colored dashed lines).

platform with four tiles interconnected by a 2x2 mesh.
Program code of computational tasks is assumed to be available in the local memory of the tiles. PEs can access their local memory without interference, e.g., by a bus with time-division-multiplexing (TDM) arbitration. Consequently, local resource access conflicts do not have to be considered. To enable time-triggered scheduling, all tiles of the platform share one global time base, e.g., a global clock.
Concerning the transport of data through the platform's NoC, we assume a packet switching protocol [1]. Further, communication between tiles is assumed to be asynchronous.

B. Application Model

An application is represented as a directed bipartite graph $g_A = (\mathbf{Q}, \mathbf{E}_a)$ (see Fig. 2). The set of nodes $\mathbf{Q} = \mathbf{T} \cup \mathbf{D}$ is the union of the set of computational task nodes $\mathbf{T}$ and the set of data entity nodes $\mathbf{D}$. A *data entity* $d \in \mathbf{D}$ holds exactly one value, e.g., of type integer. In Fig. 1, spatial bindings are illustrated by curved arrows. Computational tasks t_i are bound to PEs and data entities d_i to memories.
Directed edges $e_i \in \mathbf{E}_a \subseteq (\mathbf{T} \times \mathbf{D}) \cup (\mathbf{D} \times \mathbf{T})$ specify read and write relations between computational tasks and data entities. We assume register semantics, i.e., every write access to a data entity overwrites the previously stored information.
To integrate routing decisions for read and write transactions, we introduce *communication tasks* $c_i \in \mathbf{C}$. A bijective function $j : \mathbf{E}_a \rightarrow \mathbf{C}$ assigns each edge $e_i \in \mathbf{E}_a$ exactly one communication task $c_i \in \mathbf{C}$ from the set of communication tasks $\mathbf{C}$. For all non-local accesses to a data entity, the associated communication task has to be routed on the NoC. In Fig. 1, the decisions about the multi-hop routes of communication tasks are illustrated as colored dashed lines.
The payload of each communication task equals the size s_i of each data entity $d_i \in \mathbf{D}$. For the sake of clarity, we assume the size of each data entity equals the system's data word size. In general, data consistency of shared data entities that deviate from this assumption has to be respected in the system model. Depending on the system design, mutual exclusion mechanism need to be considered in the scheduling of resources.
For each *computational task* $t_i \in \mathbf{T}$ we define a tuple (T_i, D_i, C_i). A computational task's relative *deadline* D_i is constraint to be less than or equal to the *period* T_i of the

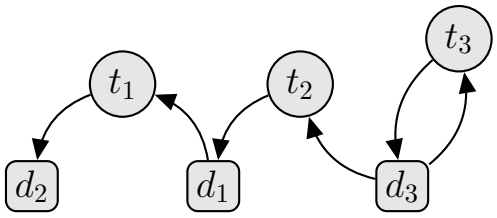

Fig. 2. Application graph of the the example presented in Fig. 1. Computational task nodes $t_i \in \mathbf{T}$ are illustrated as circles, and data entity nodes $d_i \in \mathbf{D}$ as boxes.

computational task t_i ($D_i \leq T_i$). The decision making process has to guarantee that the timing constraints of all computational tasks are satisfied, i.e., all computational tasks access all their associated data entities and process the data once per activation without violation of their relative deadline.
The *worst-case local execution time* (WCLET) of computational task t_i is denoted as C_i. The notion of local execution time is one solution to allow computation of tight WCETs for systems with interference on shared resources, as introduced in the following.

Execution semantics. To decouple shared memory communication from computation, each computational task executes in three phases: an input-phase (*I-phase*), a processing-phase (*P-phase*) and an output-phase (*O-phase*), as depict in Fig. 3. Recall that each tile includes local memory. In the *I*-phase, the computational task t_i acquires a local copy of all data entities associated with a read access. The local copies of the input data are processed in the *P*-phase. Finally, in the *O*-phase, results of the computation are written to data entities associated with a write access.
The *IPO*-semantic is a concept commonly implemented in automotive systems [11], [12]. Consequently, the WCLET C_i is defined as the maximal execution time of the *P*-phase. According to the platform assumptions, processing the local copies of data entities is not contended and program code is assumed to be locally available, too. Thus, adequate tight WCLET for each computational task can be computed using state-of-the-art WCET analyzers, e.g. [8]. Note that the *I*-phase or *O*-phase can be skipped if data is already available locally. While the above *IPO*-semantic enables us to derive a tight WCLET, interference on shared resources in the *I*- and *O*-phase needs to be addressed separately.

C. Time-Triggered Scheduling

To avoid contention on shared resources that results in non-deterministic latencies in the *I*- and *O*-phase, we compute a time-triggered schedule that guarantees contention-free access to shared resources. As an advantage of this approach, non-determinism in the termination of a computational task t_i before its WCLET C_i, which is common during runtime, does not influence the communication behavior of other computational tasks.
For all non-local read and write transactions the time triggered schedule provides a designated point in time each transaction has to be started to ensure contention-free access to shared resources. Thereby, the start of each transaction is initialized by a *push task* ρ that is executed on a PE. The execution time of the push task is bound by C_ρ. With contention-free access

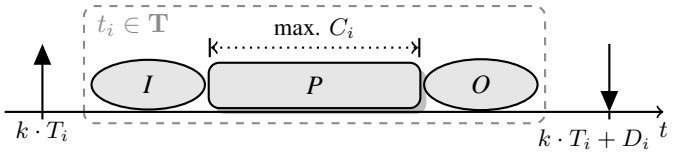

Fig. 3. Composed structure of a computational task $t_i \in \mathbf{T}$ ($k \in \mathbb{N}$). The *I*-phase fetches local copies of data entities that are processed in the *P*-phase. Non-local data entities are updated in the *O*-phase.

to shared resources, the latency of a transaction is given by the route's number of hops and the performance of the NoC. To minimize latency of read transactions, we enforce that information stored in non-local data entities is pushed to computational tasks. Hence, we distinguish two sets of push tasks: $\mathbf{D}^{\rho}_{d_i}$ and $\mathbf{T}^{\rho}_{t_i}$. The *push tasks* $\rho_{d_i} \in \mathbf{D}^{\rho}_{d_i}$ are associated with all computational tasks that access the NoC to read a non-local data entity d_i. These push tasks must be executed on the PE of the tile where the data entity is bound. On the contrary, the *push tasks* $\rho_{t_i} \in \mathbf{T}^{\rho}_{t_i}$ must be executed on the tile the computational task t_i is bound to. These push tasks initialize the transfer of a computational task's results.
The time-triggered schedule ensures that all data required to start the processing-phase (*P*-phase) of a computational task are locally available prior to its start time. Likewise, the schedule ensures that all write transactions reach their destination without violation of timing constraints.

Example. Fig. 4 illustrates the time-triggered schedule focusing on computational task t_1 of Fig. 1. The push task $\rho_{d_1} \in \mathbf{D}^{\rho}_{d_1}$ on tile 2 triggers the transaction represented as communication task c_1 at the start time $\mathbf{s}_{\rho_{d_1}}$. For the start time $\mathbf{s}_{c_1}$ of c_1 on the first subsequent interconnection of tile 2, the equation $\mathbf{s}_{c_1} = \mathbf{s}_{\rho_{d_1}} + C_\rho$ holds. The earliest start point $\mathbf{s}_{t_1}$ of computational task's t_1 *P*-phase P_{t_1} is the point in time the data, transferred by the transaction modeled as communication task c_1, is available at tile 1. Since contention-free access is guaranteed, the latency of communication task c_1, whose route consists of three hops between tile 1 and tile 2, is $3 \cdot C_c$. We denote the latency per hop of each communication task with C_c. Consequently, for the start time of the processing-phase of t_1, $\mathbf{s}_{t_1} \geq \mathbf{s}_{\rho_{d_1}} + C_\rho + 3 \cdot C_c$ holds. Push task $\rho_{d_2} \in \mathbf{T}^{\rho}_{t_1}$ starts the transaction represented as communication task c_2. With the strict separation of communication and computation by the *IPO*-semantic, $\mathbf{s}_{\rho_{d_2}} \geq \mathbf{s}_{t_1} + C_1$ holds. In the time-triggered schedule, all start times are assigned such that data entity d_2 on tile 4 is changed before the relative deadline D_1 is reached.

III. Decision Making Combining ASP and SMT

We propose a novel coupling of two CSP solvers for decision making: Employing ASP, we compute the binding of computational tasks and data entities to PEs, and the routing of communication tasks; based on these decisions, the SMT solver then computes a time-triggered schedule satisfying all timing constraints if one exists.
Due to page limitations, we cannot present the full ASP and SMT encoding. Instead, we highlight the main differences to the encodings of related work.

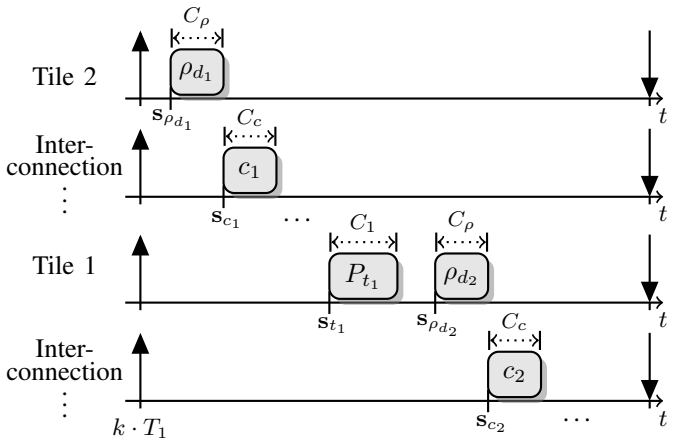

Fig. 4. The time-triggered schedule of computational task t_1 from Fig. 1 is depicted ($k \in \mathbb{N}$). Push Task $\rho_{d_1} \in \mathbf{D}^{\rho}_{d_1}$ on tile 2 triggers the transaction represented as communication task c_1 on the NoC. The processing-phase P_{t_1} of t_1 starts at the earliest with the completion of this transaction on tile 1. Push task $\rho_{d_2} \in \mathbf{T}^{\rho}_{t_1}$ initializes the write transaction of d_2 (represented as communication task c_2).

A. Spatial Binding and Routing Based on ASP

Related work on symbolic system synthesis commonly employs Pseudo-Boolean Satisfiability (PB-SAT) solving techniques to decide binding and routing in bus-based automotive controller networks [5]–[7]. In contrast to our work, the search space for routing decisions is significant smaller due to the reduced interconnection of resources.
ASP is a declarative problem solving paradigm from the area of knowledge representation and reasoning [9]. Andres et al. [3] study the problem of decision making with ASP concerning binding and routing for NoC. They show that their approach scales better for densely connected NoC, as in the case of the paper at hand, compared to equivalent PB-SAT encodings. Thereby, the scalability of ASP is enabled by the possibility to express reachability, required for multi-hop routing, in the modeling language directly [3]. The results of Andres et al. encouraged us to favor ASP over PB-SAT solving for decision making concerning binding and routing.
We modified the ASP formulation presented in [3] to fit the requirements of our system model. The ASP instance defines tiles with a fixed memory capacity and we added facts of data entities. An additional constraint in the ASP encoding ensures that no memory stores data entities beyond its capacity. The additional utilization of PEs due to push tasks $\mathbf{D}^{\rho}_{d_i}$ and $\mathbf{T}^{\rho}_{t_i}$ is respected. While we do not change the routing formulation of [3], we explicitly model the utilization of communication tasks on interconnections of the NoC.

B. Time-Triggered Scheduling Based on SMT Solving

On basis of the binding and routing decisions provided by ASP, we propose SMT solving to compute a time-triggered schedule. SMT solvers compute solutions to decision problems on logical formulae where formulae are interpreted by background theories [10]. Our SMT encoding is based on a non-preemptive scheduling formulation for an integer linear programming (ILP) solver presented in [4]. In contrast to a ILP approach, we employ a SMT solver. We are interested in computing feasible schedules fast and, if timing constraints

cannot be satisfied, we appreciate a quick response that the instance is unsatisfiable. With our scope towards satisfiability and the fact that SMT solving stems from the area of deciding the satisfiability of propositional formulas (SAT) we decided to implement a SMT solver into our approach.

The authors of [4] propose time-triggered scheduling for the automotive bus system FlexRay. While the focus of their investigations is on electronic control unit (ECU) networks, our system model covers the characteristics of applications implemented in a single many-core ECU with NoC. The proposed ILP encoding of [4] is altered to match our model and adapted for SMT solving. Especially the formulation of timing constraints is modified to comply with the requirements of computational tasks in our application model. Push tasks are scheduled on PEs together with computational tasks for non-local data access. Further, each communication tasks is synchronized with its adjacent push task and scheduled on all interconnections of its route on the NoC.

IV. Experimental Results

To demonstrate the applicability of our approach, we evaluated the combined ASP and SMT decision making method based on three early case studies. The case studies were derived from a real-world automotive ECU application. All experiments were performed on a 3.2GHz Intel i7-960 processor with 12GB RAM. We used the ASP grounder `gringo` (version 3.0.5; [13]) and the ASP solver `clasp` (version 2.1.3; [14]) with the same (sequential) search strategy as in [3]. For SMT solving, we used the `Z3` solver (version 4.3.0; [15]).

The case studies vary in their number of computational tasks $|\mathbf{T}|$, number of data entities $|\mathbf{D}|$ and number of communication tasks $|\mathbf{C}|$ (cf. Table I). Periods T_i of computational tasks $t_i \in \mathbf{T}$ range between 1ms and 200ms. In the case studies, deadlines D_i equal corresponding periods ($D_i = T_i$). We selected a platform with a regular 5x5 mesh as our target many-core hardware architecture.

Table I summarizes the particular runtimes of each solver. For all three case studies, the SMT solver returned a feasible time-triggered schedule on basis of a binding and routing computed by the ASP solver. The runtimes of each solver are reasonable and demonstrate the general applicability of our approach.

Compared to our case studies, the number of computational tasks $|\mathbf{T}|_{\text{ref}_1}$ and communication tasks $|\mathbf{C}|_{\text{ref}_1}$ scheduled in the ECU network case study of [4] is considerably smaller ($|\mathbf{T}|_{\text{ref}_1} = 48$, $|\mathbf{C}|_{\text{ref}_1} = 29$). The short runtime ($\approx$ 3s) of the employed ILP solver was observed by extending an already existing time-triggered schedule of functions incrementally. While we do not implement an incremental scheduling strategy for the extensive case studies at hand, the runtimes of the SMT solver are still fair. The scheduling strategy of [4] was improved in [5] using hierarchical scheduling to enabled a concurrent optimization approach of a case study with $|\mathbf{T}|_{\text{ref}_2} = 66$ and $|\mathbf{C}|_{\text{ref}_2} = 43$.

Although it is challenging to compare the scientific findings for automotive ECU networks to the systems discussed in this paper, our preliminary results encourage further research into the presented decision making approach. While the scalability of ASP enables us to exploit densely connected many-core systems with NoC [3], the runtimes of the proposed time-triggered scheduling using SMT solving are very reasonable with respect to the size of our case studies. The time to compute time-triggered schedules might still be improved using incremental or hierarchical scheduling approaches as in [4], [5].

TABLE I. Runtimes of the investigated case studies.

	$\lvert\mathbf{T}\rvert$	$\lvert\mathbf{D}\rvert$	$\lvert\mathbf{C}\rvert$	**Runtime ASP**	**Runtime SMT**
Case study$_1$	141	127	420	3s	16s
Case study$_2$	178	190	549	4s	26s
Case study$_3$	216	222	731	7s	52s

V. Conclusion and Future Work

In this paper, we introduced a novel model for decision making concerning time-triggered automotive many-core systems with NoC. The presented application model explicitly includes shared data entities with associated read and write accesses of computational tasks. The *IPO*-semantic separates shared memory communication from computation and, in combination with a time-triggered schedule, enables contention-free access to the shared resources of a NoC. Thereby, push tasks ensure that communication is initialized at the predefined points in time. On basis of our system model, we proposed a novel combination of ASP and SMT solving for system-level decision making. Early results of real-world case studies demonstrate the applicability of our proposed approach.

Inspired by the approaches of [5]–[7], we plan to combine the ASP and SMT solver more closely to prune the search space based on the analysis of infeasible solutions. With this extension to our approach, we intend to evaluate our system model in a concurrent optimization framework exploiting the advantages of ASP and SMT solving.

References

[1] T. Bjerregaard and S. Mahadevan, "A Survey of Research and Practices of Network-on-Chip," *ACM Comput. Surv.*, vol. 38, no. 1, 2006.

[2] A. Gerstlauer, C. Haubelt, A. D. Pimentel, T. Stefanov, D. D. Gajski, and J. Teich, "Electronic System-Level Synthesis Methodologies," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1517–1530, 2009.

[3] B. Andres et al. , "Symbolic System Synthesis Using Answer Set Programming," in *Proc. of LPNMR*, 2013, pp. 79–91.

[4] M. Lukasiewycz, R. Schneider, D. Goswami, and S. Chakraborty, "Modular Scheduling of Distributed Heterogeneous Time-Triggered Automotive Systems," in *Proc. of ASP-DAC*, 2012, pp. 665–670.

[5] M. Lukasiewycz and S. Chakraborty, "Concurrent Architecture and Schedule Optimization of Time-Triggered Automotive Systems," in *Proc. of CODES+ISSS*, 2012, pp. 383–392.

[6] F. Reimann, M. Glaß, C. Haubelt, M. Eberl, and J. Teich, "Improving Platform-Based System Synthesis by Satisfiability Modulo Theories Solving," in *Proc. of CODES+ISSS*, 2010, pp. 135–144.

[7] F. Reimann, M. Lukasiewycz, M. Glaß, C. Haubelt, and J. Teich, "Symbolic System Synthesis in the Presence of Stringent Real-Time Constraints," in *Proc. of DAC*, 2011, pp. 393–398.

[8] aiT: Worst-Case Execution Time Analyzers, AbsInt Angewandte Informatik GmbH, http://www.absint.com/ait/.

[9] C. Baral, *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

[10] C. W. Barrett et al. ,"Satisfiability Modulo Theories," in *Handbook of Satisfiability*, 2009, vol. 185, pp. 825–885.

[11] AUTOSAR: Specification of RTE, Version 3.3.0, 2013.

[12] S. Poledna, "Optimizing Interprocess Communication for Embedded Real-Time Systems," in *Proc. of RTSS*, 1996, pp. 311–320.

[13] M. Gebser, R. Kaminski, A. König, and T. Schaub, "Advances in *gringo* Series 3," in *Proc. of LPNMR*, 2011, pp. 345–351.

[14] M. Gebser, B. Kaufmann, and T. Schaub, "Conflict-Driven Answer Set Solving: From Theory to Practice," *Artificial Intelligence*, vol. 187, pp. 52–89, 2012.

[15] L. de Moura and N. Bjørner, "Z3: An Efficient SMT Solver," in *Proc. of TACAS*, 2008, pp. 337–340.