Enhancing Symbolic System Synthesis through ASPmT with Partial Assignment Evaluation

Kai Neubauer, Christian Haubelt
University of Rostock
Rostock, Germany
{kai.neubauer, christian.haubelt}@uni-rostock.de

Philipp Wanko, Torsten Schaub
University of Potsdam
Potsdam, Germany
{wanko, torsten}@cs.uni-potsdam.de

***Abstract*—The design of embedded systems is becoming continuously more complex such that efficient system-level design methods are becoming crucial. Recently, combined *Answer Set Programming* (ASP) and *Quantifier Free Integer Difference Logic* (QF–IDL) solving has been shown to be a promising approach in system synthesis. However, this approach still has several restrictions limiting its applicability. In the paper at hand, we propose a novel ASP modulo Theories (*ASPmT*) system synthesis approach, which (i) supports more sophisticated system models, (ii) tightly integrates the QF–IDL solving into the ASP solving, and (iii) makes use of partial assignment checking. As a result, more realistic systems are considered and an early exclusion of infeasible solutions improves the entire system synthesis.**

I. Introduction and Related Work

With the ever growing demand for highly complex embedded systems in both industrial and consumer environments, efficient design techniques gain progressively more importance. To satisfy the requested productivity constraints, the abstraction of embedded systems design has been raised to the electronic system level. In system synthesis, a task-level behavioral description is transformed into a structural representation considering certain constraints such as available computational (CPUs, DSPs, hardware accelerators) and communication (buses, routers) resources as well as extra-functional requirements [1].

To perform a holistic synthesis, resources have to be selected from an architectural template, computational tasks and messages have to be mapped onto computational resources and routed over the allocated communication infrastructure, respectively, and finally, a schedule has to be determined that complies with given timing constraints like latency or throughput requirements.

In order to cope with the ever increasing complexity of functional and non-functional requirements of embedded systems, symbolic techniques have traditionally been employed in embedded system synthesis since the early 2000s, e.g. [2], [3], [4]. While the encoding of feasible mapping and routing decisions into Boolean formulas led to efficient synthesis frameworks by leveraging enhancements in state-of-the-art Boolean satisfiability (SAT) solvers, numerical and non-linear problems (such as scheduling) are not easily representable in Boolean logic. As a consequence, SMT-based techniques, i.e. the combination of Boolean logic (traditionally SAT) and variant background theories, have been developed in the domain of embedded systems synthesis [5], [6], [7], [8], [9]. In one of the first works on SMT-based approaches, Reimann et al. [6] integrate a background theory solver to evaluate partial assignments of an underlying SAT-solver. In contrast to the work at hand, the authors do not consider scheduling decisions. In [10], they extend their approach to constructing schedules though only consider complete assignments, i.e. mapping and routing is already completed.

The authors of [9] present another SMT-based method for scheduling analysis in a background theory. Similar to the proposed approach in this paper, they utilize *quantifier free integer difference logic* (QF–IDL) as background theory to identify valid schedules. The main advantages of QF–IDL are its decidability in polynomial time [11] and the possibility to leverage the solving process to directly analyze and propagate observed conflicts. Yet, the work of [9] does not support cyclic or periodic applications nor heterogeneous architectures whereas our approach considers a much more sophisticated system model allowing us to synthesize cyclic and periodic behavior.

The work of Andres and Biewer et al. in [7] and [8] is the most related to our approach. They suggest to replace the SAT-solver with an *answer set programming* (ASP) solver as it has been shown to decide routing options more efficiently. To analyze timing constraints, they use separate ASP and QF–IDL-based SMT-solvers which communicate indicator variables through a shared text file. Thus, in this configuration, an evaluation of partial assignments is not supported. Furthermore, the system model considered in [7] and [8] is very restrictive. It only allows the representation of simple linear applications where deadlines of tasks have to be smaller than their corresponding periods. In combination with a homogeneous hardware architecture and unconstrained mapping options, the authors are able to exploit symmetries in the synthesis problem and use advanced heuristics which simplifies the solving accordingly.

Contributions: In this paper, we first present a mathematical formulation that allows us to describe more sophisticated system models for streaming applications on heterogeneous hardware architectures compared to previous work. In particular, this includes deadline-constrained periodic and cyclic applications. Second, we integrate the background theory *QF–IDL* directly into the state-of-the-art ASP-Solver *clingo* [12] which results in a much tighter coupling between background and foreground theories. Third, individual propagators check extra-functional constraints early in the decision process by utilizing *partial assignment checking*. Thus, large infeasible areas of design points can be excluded from the search space more efficiently.

Paper organization: In Sec. II, a detailed presentation of the underlying specification model is given. The proposed synthesis framework including binding, routing, and scheduling encodings as well as first results are presented in Sec. III. Finally, Sec. IV concludes this work and discusses future challenges.

II. Specification Model

The emphasis of this work is placed on the system synthesis of embedded systems. In this section, a mathematical description of the graph-based behavioral specification model consisting of deadline constrained periodic applications and a heterogeneous architecture template is presented.

Application: The set of applications A is composed of independent applications A_i. Each application A_i is defined as a directed graph encoded by the quintuple $A_i = (V_A^i, E_A^i, P_i, D_i, s_i)$. Here, the vertices $V_A^i = T_i \cup C_i$ represent the union of computational tasks T_i and communication messages C_i, the edges $E_A^i \subseteq T_i \times C_i \cup C_i \times T_i$ represent dependencies between tasks and messages. Each message $c \in C_i$ has exactly one predecessor task $pre(c) = t_x$ with $(t_x, c) \in E_A^i$ and one successor task $succ(c) = t_y$ with $(c, t_y) \in E_A^i$, i.e. interprocess communication is characterized in a point-to-point fashion. A period $P_i \in \mathbb{N}$ determines the time after which the execution is repeated, an absolute deadline $D_i \in \mathbb{N}$ sets the maximum latency, and the function $s_i : C_i \mapsto \mathbb{N}$ assigns an index delay to each message. That is, the message c sent by task $t_x = pre(c)$ in iteration m must arrive at task $t_y = succ(c)$ at the latest in iteration $m + s_i$. Note that cyclic graphs are only valid if the sum of index delays along a path $p = \langle t_x, c_y, ..., c_z, t_x \rangle$ is greater than zero:

$$\forall p \in \{\langle t_x, c_y, ..., c_z, t_x \rangle \mid (t_x, c_y), ..., (c_z, t_x) \in E_A^i\} : \sum_{c_k \in p} s(c_k) > 0.$$

In Fig. 1, two sample applications A_1 and A_2 are shown on the left and right, respectively:

$$A_1 = (\{t_1, t_2, t_3, t_4\} \cup \{c_1, c_2, c_3\}, E_A^1, 7, 12, \{c_x \mapsto 0 \mid x = 1, 2, 3\})$$
$$A_2 = (\{t_5, t_6\} \cup \{c_4, c_5\}, E_A^2, 10, 8, \{c_4 \mapsto 0, c_5 \mapsto 1\}).$$

The cycle $\langle t_5, c_4, t_6, c_5, t_5 \rangle$ in application A_2 is validated by the index delay $s(c_5) = 1$, indicating that the message c_5 sent by task t_6 must arrive at task t_5 in the next iteration.

Architecture Template: The architecture, or *platform* template is similarly modeled as a directed graph $P = (V_P, E_P, D_P)$. The set of vertices $V_P = R_t \cup R_{rsu}$ is partitioned into tiles R_t, containing computational resources and router switching units (RSU) R_{rsu}. Similar to [8], each RSU internally consists of independent crossbar switches and corresponding input buffers to allow concurrent routing of messages. That is, two (or more) messages may be routed simultaneously over one router if they have different destinations. A directed edge $l \in E_P \subseteq V_P \times V_P$ models a link between two resources. Finally, the routing delay $D_P \in \mathbb{N}$ determines the time for each message to get routed over one router. A (simplified) platform template is depicted in the center of Fig. 1. It contains four tiles $R_t = \{r_1, ..., r_4\}$ and four RSUs $R_{rsu} = \{rsu_1, ..., rsu_4\}$. Each tile is connected to one RSU whereas the RSUs are arranged and interconnected in a regular mesh topology. Note that bidirectional arrows represent two separate links. For example, the connection between r_1 and rsu_1 represents the directed edges $l_1 = (r_1, rsu_1)$ and $l_2 = (rsu_1, r_1)$.

Specification Model: The specification connects the set of applications and the platform template. Formally, it is defined as the quadruple $S = (A, P, M, e)$. Here, A, P and M conform to a set of applications, a platform template and a set of mapping edges, respectively. A mapping edge $m = (t, r) \in M \subseteq T \times R_t$ indicates that task $t \in T$ may be executed on resource $r \in R_t$.

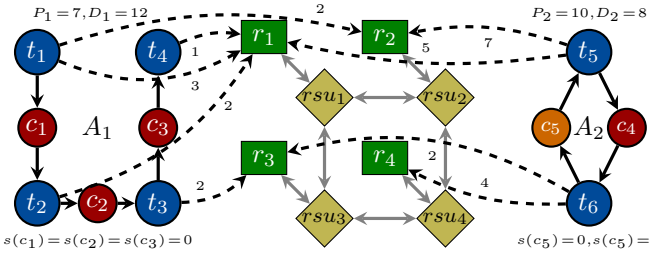

Figure 1. Example specification model containing two applications A_1 (left) and A_2 (right), an architecture template implementing an NoC with four computational resources and four RSUs (center), and nine mapping edges

The function $e : M \mapsto \mathbb{N}$ assigns a worst case execution time (WCET) to each mapping edge $m \in M$. Note that specifying more than one mapping edge m per task with different WCETs permits the modeling of heterogeneous architectures. In Fig. 1, the mapping edges are displayed as dashed curved arrows. For example, the task t_1 may be executed on tile r_1 and r_2 with worst case execution times of 3 and 2, respectively.

III. Synthesis Framework

In this section, we present a framework capable of finding a feasible implementation, i.e. binding, routing, and scheduling, for a given system specification S. Here, we consider static binding and routing. That is, tasks are mapped onto and messages are routed over the same resources in each iteration. Furthermore, a static non-preemptive periodic scheduling is assumed, i.e. the start time of each task instance (job) is determined at design time and a job completes execution before another job is started, resulting in a higher analyzability. Note that we only have to determine the start time of first occurrence of each task instance or communication hop[1] (job). Start times in subsequent iterations are derived from the period P_i of the corresponding applications.

A. Overview

Our proposed system synthesis approach combines Boolean constraint solving, in our case ASP, that covers binding and routing, and the background theory QF–IDL which encodes the timing constraints for scheduling. An overview of the system architecture is given in Figure 2. In contrast to previous work using ASP, the background theory is tightly integrated in the solving process. This is enabled by the current release of the ASP solver *clingo* 5 [12] that allows for the definition of arbitrary theory languages and easy integration of custom theory propagators.

The architecture provides three key advantages: First, the system detects invalid schedules for *partial* bindings and routings and directly learns infeasible areas of the design space. Second, an additional *post propagator* enables us to exclude cyclic schedules where jobs might overlap in later periods without unfolding tasks over several iterations. Third, only one encoding is necessary for binding, routing and scheduling allowing for a more succinct and *elaboration tolerant* formulation.

As is common in ASP, we represent the synthesis problem instance $S = (A, P, M, e)$ as a set of facts that is then combined with a general problem encoding to obtain a feasible implementation. The fact format is similar to [13], yet extended with facts to fit our specification model. A solution to the synthesis problem instance is comprised of the binding and routing, decided by the Boolean

[1] In the following, both task instances and communication hops are called jobs.

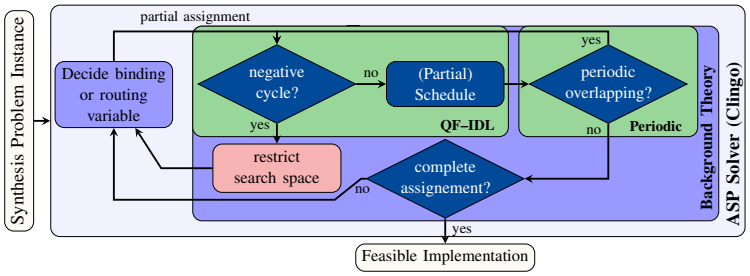

Figure 2. Architectural overview of the proposed synthesis framework.

constraint solver and an assignment of start times for each job derived by two individual background theory propagators.

B. Binding and Routing

The encoding for binding and routing is derived from [13]. Among the possible mappings $(t,r) \in M$, for each task t, exactly one mapping is chosen, denoted by $t \to r$. For every communication message c, a route is decided recursively from the receiver to the sender. We abbreviate $(t_1,c),\ (c,t_2) \in E_A^i$ by $t_1 \xrightarrow{c} t_2$, and $r_1 \xrightarrow{c} r_2$ denotes that communication c is routed over a link $(r_1,r_2) \in E_P$. The routing encoding is slightly adapted to include the index delay and to respect point-to-point communication. In the following, $\mathcal{B} \subseteq M$ and $\mathcal{R} \subseteq \{r_x \xrightarrow{c_k} r_y \,|\, r_x,r_y \in E_P, c_k \in C\}$ denote a (partial) binding and routing, respectively.

C. Scheduling

As ASP does not scale for numerical and non-linear constraints, we employ QF–IDL in the background theory that deals with constraints of form $x-y \leq k$, with variables $x,y \in \mathbb{Z}$ and constant $k \in \mathbb{Z}$, to encode timing constraints. To incorporate QF–IDL into ASP, we rely on the stateful propagator presented in [12]. The set of QF–IDL constraints $\mathbb{C}$ is formulated as follows:

$$\mathbb{C} = \{0 - \tau(t) \leq 0 \,|\, t \in T_i, (T_i \cup C_i, E_A^i, P_i, D_i, s_i) \in A\} \quad (1)$$

$$\cup \{\tau(t) - 0 \leq D_i - e((t,r)) \,|\, t \to r \in \mathcal{B}\} \quad (2)$$

$$\cup \{\tau(t_x) - \tau(t_y) \leq -e((t_x,r)) + s(c) \cdot P \,|\, t_x \xrightarrow{c} t_y, \{t_x \to r, t_y \to r\} \subseteq \mathcal{B}\} \quad (3)$$

$$\cup \{\tau(t_x) - \tau(c^r) \leq -e((t_x,r)) \,|\, t_x \xrightarrow{c} t_y,\ t_x \to r \in \mathcal{B}, t_y \to r \notin \mathcal{B}\} \quad (4)$$

$$\cup \{\tau(c^{r_x}) - \tau(c^{r_y}) \leq -D_P \,|\, r_x \xrightarrow{c} r_y \in \mathcal{R}\} \quad (5)$$

$$\cup \{\tau(c^{r_x}) - \tau(t_y) \leq -D_P + s(c) \cdot P \,|\, \{t_x \xrightarrow{c} t_y, r_x \xrightarrow{c} r_y\} \subseteq \mathcal{R}, t_y \to r_y \in \mathcal{B}\} \quad (6)$$

$$\cup \{\tau(t_x) - \tau(t_y) \leq -e((t_x,r)) \,|\, \{t_x \to r, t_y \to r\} \subseteq \mathcal{B}, t_x \succ t_y\} \quad (7)$$

$$\cup \{\tau(c_x^{r_x}) - \tau(c_y^{r_x}) \leq -D_P \,|\, \{r_x \xrightarrow{c_x} r_y, r_x \xrightarrow{c_y} r_y\} \subseteq \mathcal{R}, c_x \succ c_y\} \quad (8)$$

Here, $\tau(t)$ indicates the start time of task instance t. Analogously, $\tau(c^r)$ encodes the time point a communication message c is sent from resource r. The set $\mathbb{C}$ of QF–IDL constraints that are true during the solving process depends on the instance S, the current (partial) binding $\mathcal{B}$ and routing $\mathcal{R}$ as well as a strict, partial order between tasks $\succ$. We decide the order $\succ$ between two tasks in the ASP solver, i.e. either $t_x \succ t_y$ or $t_y \succ t_x$ holds. Integrity constraints ensure that no cyclic orders such as $(t_x \succ t_y,\ t_y \succ t_z,\ t_z \succ t_x)$ are induced. We extend the order to apply for communications $c_x,\ c_y$, such that $c_x \succ c_y$ iff $(t_x,c_x) \in E_A^i, (t_y,c_y) \in E_A^j$ and $t_x \succ t_y$. Considering multiple applications A_i with different periods P_i, we have to determine the hyper period P_H by calculating the least common multiple of all periods: $P_H = \mathrm{lcm}_{A_i \in A}(P_i)$. The adapted deadline $D_{H,i}$ for each application A_i is additionally calculated as: $D_{H,i} = P_H + (D_i - P_i)$. Intuitively, constraints (1) and (2) make sure that task instances in the first iteration are executed in between time point 0 and their deadline D_i, constraints (3) to (6) schedule communications $t_x \xrightarrow{c} t_y$ depending on the binding of t_x, t_y and routing of c, and (7) and (8) resolve conflicts between independent tasks and communications mapped to the same resource according to $\succ$.

As visualized in the green boxes in Fig. 2, the validity of $\mathbb{C}$ is tested in a two part process.

1) Check for consistency of $\mathbb{C}$: The QF–IDL propagator validates consistency by creating the constraint graph $G_\mathbb{C}$, where all start times $\tau(t)$ and $\tau(c^r)$ occurring in constraints in $\mathbb{C}$ form the nodes. Additionally, for each inequality $x - y \leq k \in \mathbb{C}$ there exists an edge in $G_\mathbb{C}$ from x to y with weight k. The propagator now intends to find the shortest path to each node, which results in the negated earliest possible start time τ for the corresponding node. If a negative cycle is detected in $G_\mathbb{C}$, no schedule can be determined. Hence, the variables representing the cycle are added as a conflict clause and the ASP solver has to backtrack. Otherwise, a valid partial assignment $v : V \to \mathbb{Z}$ is found. Note that the consistency is checked on partial assignments. That is, if a negative cycle is found, a large area of the the search space can be pruned early.

2) Check for periodic overlapping: Even though the constraint Graph $G_\mathbb{C}$ is now consistent, jobs that are bound to the same resource might overlap with other jobs in later iterations whenever the deadline $D_{H,i}$ is greater than the hyper period P_H. This is checked by a dedicated propagator. Each job that is executed later than the first period is identified as a candidate for overlapping. Now, the execution time span of all other jobs on the same resources is projected to the iterations of the candidates in question. If the time span overlaps with the candidates time span, a timing constraints has to be added to serialize the execution on the same resource in that iteration. Let $t_x \in A_i$ be the candidate and $t_y \in A_j$ be a job mapped to the same resource r, for which a conflict has been found. If $t_x \succ t_y$, the constraint $\tau(t_x) - \tau(t_y) \leq -e((t_x,r)) + i_r \cdot P_j$ is added to $\mathbb{C}$, where i_r is the difference between the iteration the conflict was found in and the first iteration t_y is executed in. This forces t_x to be executed before t_y in the iteration where an overlapping was found. Analogously, if $t_y \succ t_x$, the constraint $\tau(t_y) - \tau(t_x) \leq -e((t_y,r)) - i_r \cdot P_j$ is added to $\mathbb{C}$, forcing t_x to be executed after t_y. In case no overlapping has been found, v is a valid partial assignment. If $\mathcal{B}$ and $\mathcal{R}$ are total assignments, v, $\mathcal{B}$, and $\mathcal{R}$ form a feasible implementation. Otherwise, the ASP solver proceeds with binding and routing. If overlapping was detected, we return to 1) with the extended set of constraints $\mathbb{C}$.

As an example for the scheduling process, we consider application A_1 from Figure 1. For the sake of brevity, we assume the communication between tasks to be instantaneous, i.e. the execution of a job t_x is only dependent on its direct predecessor job(s). Furthermore, we presume the tasks t_1,t_2 and t_4 to be mapped to resource r_1 and t_3 to be mapped to r_3. In the first step, a constraint graph $G_\mathbb{C}$ is built upon the dependencies

Table I. EXPERIMENTAL RESULTS OF THE RANDOMLY GENERATED CASE STUDIES

|R|	|T|	|C|	Partial Assignments				Full Assignments			
			Choices	Conflict Length (Conflicts)	Runtime Overall	Runtime QF–IDL	Choices	Conflict Length (Conflicts)	Runtime Overall	Runtime QF–IDL
2×2	28	32	**3364**	40.6 (1253)	**12.70 s**	11.93 s	135692	**39.5 (1235)**	14.69 s	2.65 s
2×2	20	44	**11680**	**15.8 (927)**	**87.17 s**	85.19 s	482870	22.9 (5272)	173.08 s	53.06 s
3×3	18	18	**181606**	**26.2 (109431)**	**399.37 s**	367.56 s	17445197	27.1 (119620)	994.80 s	187.35 s
3×3	45	50	**47439**	**37.7** (1343)	388.64 s	380.17 s	135546	48.2 (**495**)	**91.47 s**	10.02 s
4×4	48	58	**20851**	53.5 (**549**)	355.95 s	347.17 s	287231	**45.8** (2256)	**187.56 s**	29.82 s
4×4 (UNSAT)	48	54	**235214**	25.5 (119953)	**1345.67 s**	1231.92 s	18346301	**20.4 (29024)**	T/O	

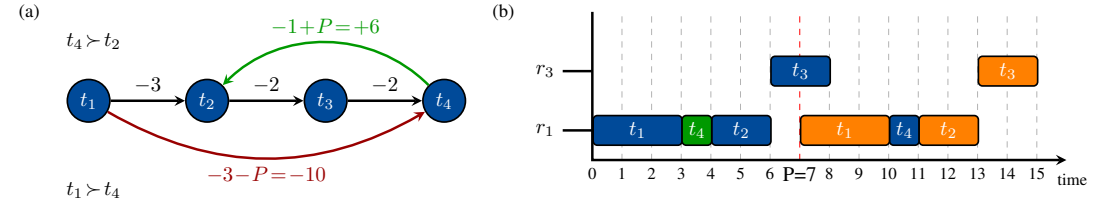

Figure 3. (a) Constraint graph $G_{\mathbb{C}}$ based on application A_1 and the order $\succ$ decided by the ASP-solver. (b) The corresponding schedule (snapshot) visualized by a Gantt chart. The variant colors represent different iterations: blue indicates the current, orange the next, and green the previous iteration.

and execution times. This is represented by the black arrows in Fig. 3 (a). By calculating the shortest path to each node of $G_{\mathbb{C}}$, we get the initial start times for the first iteration: $\tau(t_1)=0$, $\tau(t_2)=3$, $\tau(t_3)=5$, $\tau(t_4)=7$. As the period P_1 of A_1 is equal to 7, the post propagator detects periodic overlapping between task t_1 and t_4. As a consequence, they have to be serialized. Considering $t_1 \succ t_4$, the constraint $t_1 - t_4 \leq -e(t_1) - i_r \cdot P_1 = t_1 - t_4 \leq -10$ is added with $i_r = 1$ since the overlap occurs one iteration apart (red arrow). Now, t_4 overlaps with t_2 in the second iteration and, assuming $t_4 \succ t_2$, the constraint $t_4 - t_2 \leq -e(t_4) + i_r \cdot P_1 = t_4 - t_2 \leq 6$ is added, forcing t_2 to be executed one time step later (green arrow). Finally, all overlaps have been resolved and $G_{\mathbb{C}}$ in Fig. 3 (a) represents a valid schedule which is visualized by a Gantt chart in Fig. 3 (b). Note that $t_4 \succ t_1$ would introduce a constraint forming a negative cycle in the constraint graph $G_{\mathbb{C}}$.

D. Experimental Results

Various specification instances consisting of a set of series-parallel applications as well as an architectural template implementing a regular, heterogeneous NoC have been generated randomly to test the proposed approach. All test cases were executed on a Core i7-5600U with 16 GiB RAM and a timeout of 1800 s. As shown in Tab. I, our experiments indicate that ASPmT using partial assignment evaluation leverages promising results. In particular, the number of choices necessary to find a valid implementation when utilizing partial assignment checking is approximately one order of magnitude smaller compared to the evaluation of full assignments only. While the number of conflicts and average conflict length are comparable between both approaches, it shows that the search space is pruned more efficiently.

Nevertheless, the reduction of necessary choices does not correlate with the overall runtime for all test cases. As seen in Tab. I, most of the time is spend in the background propagators when using partial assignment checking. As this is still work in progress, the performance of said propagators are subject to implementational improvements in future iterations.

IV. CONCLUSION AND FUTURE WORK

In this paper, we presented a novel ASPmT-based approach towards symbolic system synthesis that supports a more expressive system model compared to previous work. Especially, the proposed approach allows for the description of deadline-constrained periodic applications as well as heterogeneous hardware architectures. Furthermore, the tight integration of QF–IDL into the ASP solver allows us to evaluate partial assignments in order to prune the search space more efficiently. First experimental results showed the general applicability of our approach.

In the future, we aim to improve the performance of used propagators for example by using lazy variable creation or incorporating domain-specific heuristics. Since our ASPmT framework encompasses all relevant information, the model is easily adaptable and features may be added comfortably. Namely, we will extend the framework towards design space exploration including multi-objective optimization techniques.

ACKNOWLEDGMENT

This work was funded by the German Science Foundation (DFG) under grant 4463/4-1.

REFERENCES

[1] A. Gerstlauer et al. Electronic System-Level Synthesis Methodologies. *IEEE TCAD*, 28(10):1517–1530, 2009.

[2] C. Haubelt et al. SAT-based techniques in system synthesis. In *Proc. of DATE*, pages 1168–1169, 2003.

[3] M. Lukasiewycz et al. Combined system synthesis and communication architecture exploration for MPSoCs. In *Proc. of DATE*, pages 472–477, 2009.

[4] M. Lukasiewycz et al. Modular scheduling of distributed heterogeneous time-triggered automotive systems. In *Proc. of ASP-DAC*, pages 665–670, 2012.

[5] N. Satish et al. A decomposition-based constraint optimization approach for statically scheduling task graphs with communication delays to multiprocessors. In *Proc. of DATE*, pages 1–6, 2007.

[6] F. Reimann et al. Improving platform-based system synthesis by satisfiability modulo theories solving. In *Proc. of CODES/ISSS*, pages 135–144, 2010.

[7] B. Andres et al. Improving coordinated smt-based system synthesis by utilizing domain-specific heuristics. In *Proc. of LPNMR*, pages 55–68, 2015.

[8] A. Biewer et al. A symbolic system synthesis approach for hard real-time systems based on coordinated SMT-solving. In *Proc. of DATE*, pages 357–362, 2015.

[9] W. Liu et al. Satisfiability Modulo Graph Theory for Task Mapping and Scheduling on Multiprocessor Systems. *IEEE TPDS*, 22(8):1382–1389, 2011.

[10] F. Reimann et al. Symbolic system synthesis in the presence of stringent real-time constraints. In *Proc. of DAC*, pages 393–398, 2011.

[11] S. Roberto. Lazy satisfiability modulo theories. *Journal on Satisfiability, Boolean Modeling and Computation*, 3:141–224, 2007.

[12] M. Gebser et al. Theory solving made easy with clingo 5. In *Technical Communications of ICLP*, 2016. To appear.

[13] B. Andres et al. Symbolic system synthesis using answer set programming. In *Proc. of LPNMR*, pages 79–91, 2013.