A Symbolic System Synthesis Approach for Hard Real-Time Systems Based on Coordinated SMT-Solving

Alexander Biewer[1], Benjamin Andres[2], Jens Gladigau[1], Torsten Schaub[2], Christian Haubelt[3]
[1]Corporate Sector Research, Robert Bosch GmbH, 71701 Schwieberdingen, Germany
[2]Knowledge Processing and Information Systems, University of Potsdam, 14482 Potsdam, Germany
[3]Applied Microelectronics and Computer Engineering, University of Rostock, 18119 Rostock, Germany

***Abstract*—We propose an SMT-based system synthesis approach where the logic solver performs static binding and routing while the background theory solver computes global time-triggered schedules. In contrast to previous work, we assign additional time to the logic solver in order to refine the binding and routing such that the background theory solver is more likely to find a feasible schedule within a reasonable amount of time. We show by experiments that this coordination of the two solvers results in a considerable reduction of the overall synthesis time.**

I. Introduction

With increasing system complexity due to customer demands, regulatory requirements, and migration to integrated architectures on massively parallel hardware, the inevitable task of system-level synthesis has become more and more challenging. During synthesis, the spatial binding of computational tasks to processing elements (PEs), the multi-hop routing of messages, and scheduling of tasks and messages on shared resources has to be decided.

In the domain of next generation hard real-time systems, many previously proposed synthesis approaches might fail due to the sheer complexity or the stringent timing constraints. System synthesis based on satisfiability modulo theories (SMT) has been shown to solve this problem by splitting the work among a logic (generally SAT) solver and a background theory solver ($\mathcal{T}$-solver) [1].

In the work at hand, similar to [1], the logic solver decides the binding of periodic computational tasks to the PEs of a tile-based many-core platform. Furthermore, the logic solver determines multi-hop routes of messages on the resources of the network-on-chip (NoC) interconnect of the platform. Based on these decisions, a linear arithmetic solver ($\mathcal{T}$-solver) then computes global time-triggered schedules. In order to link both solvers, indicator variables are assigned within the logic solver and used to constrain the solution space for the $\mathcal{T}$-solver. Moreover, the indicator variables are used by the logic solver to learn unschedulable implementations from the conflict analysis in the background theory.

However, it can be observed that very often the time-triggered scheduling becomes too complex and the $\mathcal{T}$-solver cannot decide within a reasonable time budget if a binding and routing (B&R) provided by the logic solver is schedulable or not. Thus, the feedback from the $\mathcal{T}$-solver to the logic solver as stated in [1] is not given. This is especially unfortunate as the logic solver is able to produce a B&R within seconds. As a remedy, we propose to assign an additional amount of time to the logic

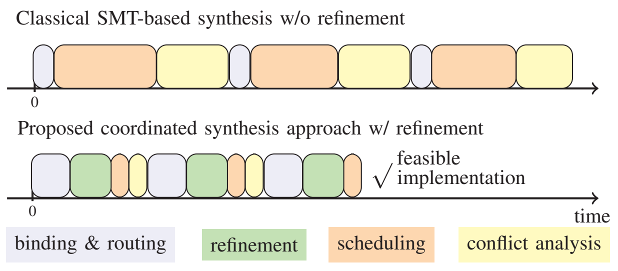
Classical SMT-based synthesis w/o refinement
0
Proposed coordinated synthesis approach w/ refinement
√ feasible implementation
0
time
binding & routing
refinement
scheduling
conflict analysis

Fig. 1. Comparison of the classical SMT-based system synthesis (without refinement) and the coordinated approach (including refinement). Additional time is assigned to the logic solver in order to refine the binding and routing. The resulting solutions are expected to be easier instances for scheduling.

solver in order to refine B&R solutions in a way that it is expected to be an easier instance for scheduling.

One way to achieve this, and done in this paper, is that the logic solver uses the additional time to perform preferably load balancing as near to fully-utilized resources make scheduling much harder. The effort of our proposed approach on the overall synthesis time is sketched in Fig. 1.

The upper part of Fig. 1 illustrates the time spent in the different stages of a classical SMT-based synthesis. While the time to derive a binding of computational tasks and routing of messages is very short, deciding the schedulability and a subsequent conflict analysis in the $\mathcal{T}$-solver is very time consuming. Conflict analysis is a common part in SMT-based system synthesis as it enables a more efficient learning of unschedulable B&R in the logic solver [2].

The lower part of Fig. 1 illustrates our proposed coordinated approach. In contrast to a system synthesis without refinement, the coordinated approach consumes more time to derive a B&R in the logic solver. This is due to the fact that the additional formulation of the refinement's objectives results in harder instances for the logic solver (see Section V and Section VI). Further, Fig. 1 depicts the additional constant time budget that is assigned to the logic solver in order to refine the B&R. The logic solver's refined B&R solutions are easier instances for the $\mathcal{T}$-solver, though the scheduling and conflict analysis takes considerably less time compared to a synthesis without refinement. As our experiments show (cf. Section VI), our proposed coordinated approach often results in significantly reduced overall synthesis times.

The remainder of this paper is organized as follows: After surveying related work, Section III provides an overview of our proposed approach. In Section IV we provide a formal problem definition and the formal system specification. Section V then summarizes the problem encoding and further concepts of our approach. Section VI reports experimental results that show the effectiveness of our coordinated SMT-based synthesis approach. Section VII concludes this paper.

II. Related Work

In order to manage increasing system complexity, symbolic system synthesis approaches utilizing pseudo-boolean satisfyability (PB-SAT) solving emerged as one solution to deal with intricate system specifications where former heuristic approaches tended to fail [3]. Thereby, even complex system specifications, e.g., regarding maximal bus load, could be handled efficiently. However, these approaches are restricted to linear constraints. Further, a binary encoding of integers leads to an exponential growth in variables. In order to handle non-linear functional constraints, e.g., non-linear timing constraints, and large numeric domains, symbolic system synthesis approaches leveraging advances from the the area of SMT [4] have become a scalable solution to deal with such intricate system specifications [1], [2], [5], [6]. There, SMT solvers decide the satisfiability of decision problems on logical formulae, where formulae are interpreted by background theories. In general, the propositional part of an SMT solver is realized by a SAT solver while a solution with respect to the background theory is computed by a so-called $\mathcal{T}$-solver.

A first SMT-based system synthesis approach is presented in [7]. In [8] the authors use a latency computation as background theory and perform optimization by a branch and bound strategy incorporated into the SMT solver. Another SMT-based approach similar to ours is proposed in [9]. The underlying platform is a time-triggered architecture. As background theory, the authors use linear arithmetics for adding worst-case execution times. However, no multi-hop communication is considered. Considering multi-hop routing, fundamental work on SMT-based symbolic system synthesis was done by Reimann et al. [1], [2]. To solve the problem of assigning speeds to resources with respect to delay, buffer and energy constraints, the SMT-based approach presented in [6] extended the work by Reimann et al. with an analysis of incomplete models. In [5], Lukasiewycz and Chakraborty present an SMT-based architecture and scheduling optimization approach. The work at hand makes use of some of the discussed concepts, e.g., parts of the scheduling encoding of this paper are based on the encoding shown in [5]. None of these approaches, however, studies the coordination between logic and $\mathcal{T}$-solver.

Andres et al. [10] show that a system synthesis using answer set programming (ASP) can have advantages in terms of scalability compared to PB-SAT approaches if the target platform is highly interconnected. The results of [10] encouraged us to implement ASP as well. However, the work by Andres et al. does not discuss synthesis with respect to background theories. While ASP including background theories has already been studied, e.g., [11], the domain specific concepts discussed in this paper cannot be used in frameworks like [11] in a straight forward manner.

III. Overview

Fig. 2 illustrates the SMT-based synthesis approach presented in this paper. On basis of the system's formal specification (cf. Section IV), the synthesis problem is solved by splitting the overall problem into the subproblem of B&R and the subsequent problem of scheduling. A formal representation of both problems that need to be solved including details on how both problems are linked via indicator variables is presented in Section V.

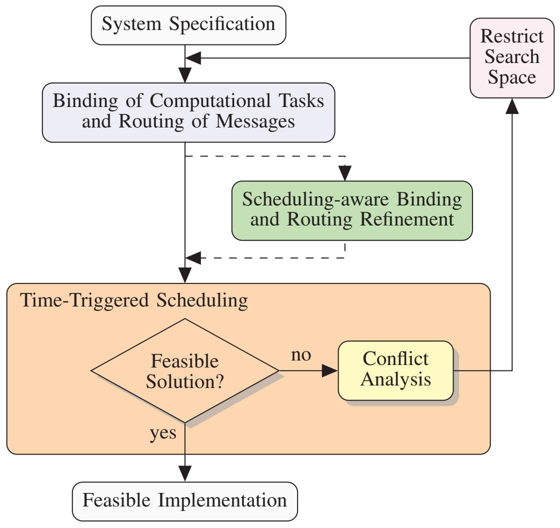
System Specification
Restrict Search Space
Binding of Computational Tasks and Routing of Messages
Scheduling-aware Binding and Routing Refinement
Time-Triggered Scheduling
Feasible Solution?
no
yes
Conflict Analysis
Feasible Implementation

Fig. 2. Overview of the proposed coordinated synthesis approach presented in this paper.

Decoupling the synthesis problem enables a more scalable symbolic synthesis for hard real-time systems [1]. While related approaches implement a SAT solver to derive a static binding of computational tasks and a static multi-hop routing for messages [1], [2], [5], the approach presented in this paper replaces the SAT solver with an answer set solver from the field of answer set programming (ASP) [12] for the same task. For the synthesis of massive-parallel platforms with NoC as discussed in this paper, an answer set solver is most suitable: Related work [10] has shown that ASP enables a more scalable determination of multi-hop routes of messages compared to SAT solving approaches for densely connected hardware architectures, such as network-on-chip (NoC).

To link the subproblem of B&R solved by the answer set solver with the subsequent scheduling problem in the $\mathcal{T}$-solver, indicator variables are introduced. In this paper, the indicator variables represent the concrete binding of each computational task to exactly one tile and the hops of messages between two interconnected resources. On basis of the indicator variable assignment, a time-triggered scheduling problem in linear arithmetic is formulated (cf. [5]) and solved by the $\mathcal{T}$-solver. If the time-triggered scheduling problem based on the assignment of the indicator variables can be solved, a feasible binding, routing and scheduling has been found; the synthesis approach stops and a feasible implementation is the result. If there is no schedule that satisfies the timing constraints of the computational tasks and messages, a conflict analysis is started. A minimal reason why no feasible schedule exists is derived in the conflict analysis. This enables a more efficient learning of unschedulable B&Rs in the logic solver [2] and is used to further increase the scalability of SMT-based synthesis [5], [6].

With the results of the conflict analysis, new constraints are formulated using the indicator variables that link the answer

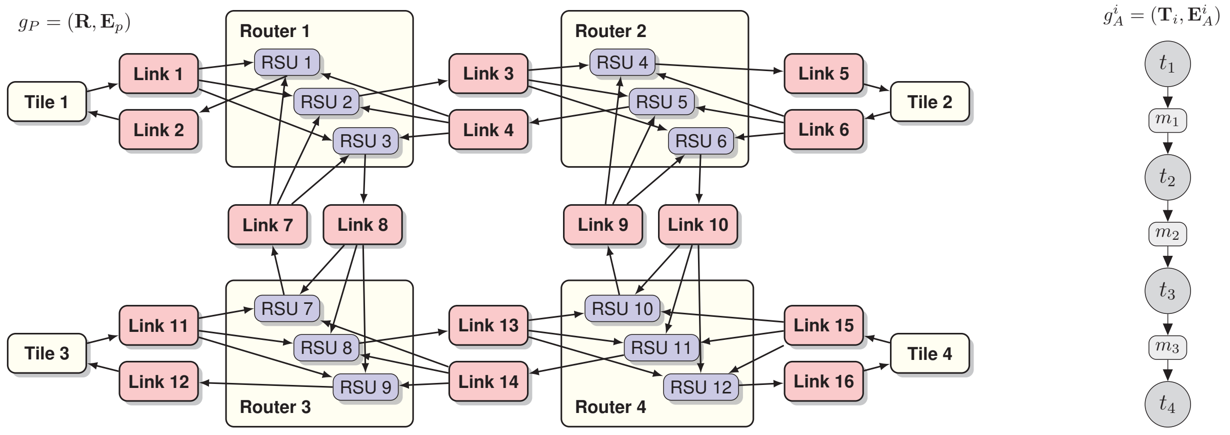

Fig. 3. Examples of a platform graph $g_P = (\mathbf{R}, \mathbf{E}_p)$ representing a 2x2 regular mesh platform and an application graph $g_A^i = (\mathbf{T}_i, \mathbf{E}_A^i)$.

set solver with the $\mathcal{T}$-solver. These new constraints reflect the minimal reason that led to the infeasible schedule and are used to restrict the search space of B&R solutions in the subsequent search in the answer set solver. This iterative coupling between the computation of B&R and the scheduling part of our approach stops if a feasible time-triggered schedule is found, as mentioned earlier. However, it could be possible that after a number of iterations no further B&R can be derived. In this case, the SMT-based approach proved that there is no feasible implementation with respect to the system specification.

A major drawback of the synthesis approach described above lies in the often very complex and time consuming scheduling in the background theory. In extreme cases, the answer set solver provides a B&R within milliseconds, whereas the scheduling takes hours. For example, this might be the case if one PE of the system is utilized close to the maximum utilization of 100%. Even worse, after a long solving time the $\mathcal{T}$-solver often proves that the given B&R cannot be scheduled. As a solution to this problem we propose an additional step (dashed lines in Fig. 2): a scheduling-aware B&R refinement. In this step, the B&R derived by the answer set solver is refined such that the schedulability of the refined B&R solution is expected to be decided within a reasonable amount of time. Roughly speaking, the refinement enables the answer set solver's awareness that there is a subsequent scheduling stage in the synthesis approach that has to be solved as well and contributes to the overall synthesis time. In our approach, we assign the answer set solver a constant time budget exclusively to optimize load balancing, the number of communication hops and resource sharing of a initially computed B&R solution. In Section VI, we will show by experiments that despite the additional time consuming refinement step, the overall synthesis time can be reduced and more complex systems can be synthesized.

IV. Problem Formulation

In this section, we formulate the synthesis problem on basis of the formal platform model and application model. The application model represents periodic hard real-time applications, e.g., from the automotive domain, and the platform model represents a tile-based many-core platform implementing an NoC.

The homogeneous tile-based many-core architecture is modeled as a directed graph $g_P = (\mathbf{R}, \mathbf{E}_p)$ (cf. Fig. 3). The nodes $\mathbf{R}$ model all possibly shared resources. A directed edge $e \in \mathbf{E}_p \subseteq (\mathbf{R} \times \mathbf{R})$ in the platform graph models a unidirectional interconnection between two resources of the hardware platform. Resources $\mathbf{R}$ can further be partitioned in tiles $\mathbf{R}_t \subseteq \mathbf{R}$, links $\mathbf{R}_l \subseteq \mathbf{R}$ and router switching units (RSUs) $\mathbf{R}_{rsu} \subseteq \mathbf{R}$. A tile implements exactly one processing element (PE) with local memory. Links $\mathbf{R}_l$ model interconnections between resources of the NoC. Since messages might share links and RSUs during their transfer on the NoC and suffer from different delays on both resources, links and RSUs of a platform are represented explicitly in the platform model. Router switching units $\mathbf{R}_{rsu}$ model the capabilities of a router's crossbar to concurrently switch between input buffers of incoming ports and output buffers of outgoing ports of a router. Note that the platform model does not represent buffers in routers explicitly. Buffers are implicit between each link-RSU interconnection and RSU-link interconnection. For the sake of clarity, we assume that the number of RSUs in a router equals the number of outgoing ports of a router. Figuratively, an RSU can then be assigned to switch different incoming buffers to the buffers of exactly one outgoing port. Following this assumption, each router in Fig. 3 can concurrently forward three incoming messages from different input buffers as long as two messages do not request the same output port.

Concerning the application model, the set of applications $\mathbf{A}$ that has to be executed on the hardware platform is assumed to consist of different applications $A_i \in \mathbf{A}$. Each application is specified by the tuple

$$A_i = (g_A^i, P_i, D_i). \tag{1}$$

An application requests to be executed periodically with the period P_i. All computation and communication of an application has to be completed within the constrained relative deadline $D_i \leq P_i$.

The graph $g_A^i = (\mathbf{T}_i, \mathbf{E}_A^i)$ is a directed acyclic bipartite graph (cf. Figure 3). For the sake of clarity, we assume in the following that each application is a connected graph. The set of nodes $\mathbf{T}_i = \mathbf{T}_i^t \cup \mathbf{T}_i^m$ is the union of the set of computational tasks of an application $\mathbf{T}_i^t$ and the set of messages of an application $\mathbf{T}_i^m$. The directed edges $\mathbf{E}_A^i \subseteq (\mathbf{T}_i^t \times \mathbf{T}_i^m) \cup (\mathbf{T}_i^m \times \mathbf{T}_i^t)$ of the graph g_A^i specify data dependencies between computational tasks and messages or vice versa.

With each computational task $t \in \mathbf{T}_i^t$ a worst-case execution time (WCET) C_t that holds for all PEs is associated. Concerning messages $m \in \mathbf{T}_i^m$, the remainder of the paper assumes that a message equals exactly one packet that is transmitted on the NoC. With this definition, the worst-case transfer time of a message $m \in \mathbf{T}_i^m$ on a resource $r \in \mathbf{R} \setminus \mathbf{R}_t$ is defined by C_m^r.

Problem Formulation. In the design of time-triggered real-time systems, system synthesis has to compute a valid implementation $I = (\mathbf{B}, \mathbf{R}_m, \mathbf{S})$ consisting of a binding

$$\mathbf{B} \subseteq \left(\bigcup_{A_i \in \mathbf{A}} \mathbf{T}_i^t\right) \times \mathbf{R}_t,$$

a routing of each message $\mathbf{R}_m \subseteq \mathbf{R} \setminus \mathbf{R}_t$ and a feasible global time-triggered schedule $\mathbf{S}$.

A binding $\mathbf{B}$ contains for each computational task $t \in \mathbf{T}_i^t$ of an application $A_i \in \mathbf{A}$ exactly one tile it is executed on. A routing $\mathbf{R}_m$ defines a set of connected resources of the NoC that are utilized during the transfer of the message on the NoC. A time-triggered schedule

$$\mathbf{S} = \{\mathbf{s}_\lambda^r \mid \lambda \in (\mathbf{T}_i^t \cup \mathbf{T}_i^m), A_i \in \mathbf{A}, r \in \mathbf{R}_m \vee (\lambda, r) \in \mathbf{B}\} \quad (2)$$

contains start times $\mathbf{s}_\lambda^r$ for each computational task on the tile it is bound to and start times for each message on the resources over which the message is routed.

V. Problem Encoding & Solver Coupling

In this section, the essential parts of our proposed problem encoding are discussed. Since our approach leverages concepts already discussed extensively in related work, we focus on a succinct survey of common features and differences between our encoding and the encodings of previous work.

B&R Encoding in ASP. To derive a B&R, we modified the ASP from [10] in the following way: We add binding constraints that ensure that no tile is utilized by computational tasks more than a fixed constant $U_{max} \leq 100$. Further, we adjust the routing encoding in [10] to our platform representation such that messages are only routed on the resource of the NoC. Additionally, constraints ensure that a router of the NoC is only visited once by each message. As a consequence, we can omit predefined mapping constraints as proposed in [10].

Refinement Encoding in ASP. To enable the additional scheduling-aware refinement step shown in Fig. 2, we utilize the `minimize`-optimization statements in the modeling language of ASP. Overall we add three optimization functions with different priorities that are respected during a lexicographical optimization. The highest priority is assigned to a simplified load balancing strategy. The solver tries to minimize a value $u \leq U_{max}$ that represents a new upper bound on the utilization of all tiles. By enabling `minimize`{u}, for a final binding $\mathbf{B}$ the following equation holds:

$$\forall r \in \mathbf{R}_t, t \in \mathbf{T}_i^t, A_i \in \mathbf{A} : \textstyle\sum_{(t,r) \in \mathbf{B}} \dfrac{C_t}{P_i} \leq u. \quad (3)$$

To bound all possible values of u, we only allow 10 evenly distributed discrete values ranging from the mean utilization of all tiles to U_{max}. Note that encoding the load balancing results in additional variables and constraints in the final ASP program in comparison to the problem encoding without refinement. Our experiments in Section VI quantify higher solving times introduced by the additional constraints.

If the load cannot be further balanced, the minimization of the overall number of hops of messages between resources of the NoC is started. Since the hops between resources are already part of the ASP encoding, no additional variables or constraints are added.

The last objective that is considered if no further communication hop reduction under the given load distribution is possible, is the minimization of the overall number of resources that are shared by different applications of the system. For this purpose, new variables have to be introduced that encode if a resource of the platform is utilized by an application. Consequently, enabling the refinement is likely to result in noticeably harder problems for large platform instances. This, again, is quantified in our experiments.

Scheduling Encoding. The scheduling encoding for the time-triggered many-core platform assumed in this paper is based on the work in [5]. The presented non-preemptive scheduling formulation for automotive networks implementing a FlexRay bus is adapted to conform with the packet-switched time-triggered NoCs of the paper at hand. One main difference between both formulations is the fact that a global time-triggered schedule can be derived by using an SMT-solver with quantifier-free integer difference logic (`QF_IDL`). Thereby, all constraints are compositions of the terms

$$\mathbf{s} - \tilde{\mathbf{s}} \lhd k \quad (4)$$

with variables $\mathbf{s}, \tilde{\mathbf{s}} \in \mathbb{Z}$, a constant $k \in \mathbb{Z}$ and relations $\lhd \in \{<, \leq, >, \geq, =\}$. As an example, a precedence constraint between two computational tasks $t, \tilde{t} \in \mathbf{T}_i^t$ in an application $A_i \in \mathbf{A}$ with the start times $\mathbf{s}_t, \mathbf{s}_{\tilde{t}} \in \mathbb{N}$ is expressed by

$$\mathbf{s}_t - \mathbf{s}_{\tilde{t}} \leq C_t. \quad (5)$$

Here, C_t is the WCET of computational task t. Note that `QF_IDL` can be decided in polynomial time. Following `QF_IDL`, all the parameters adhering to a notion of time introduced previously in Section IV, e.g., periods P_i of applications $A_i \in \mathbf{A}$, WCETs C_t of tasks or transfer times of messages C_m^r are represented as natural numbers in units of a system-wide global clock.

Hierarchical Scheduling. Similar to [5], we propose a domain specific hierarchical scheduling scheme in order to (a) decrease the time to prove that a B&R is not schedulable and (b) decrease the time consumed by conflict analysis. In our hierarchical scheduling scheme, the task of deriving a time-triggered schedule for the complete system is decomposed into smaller subproblems that have to be scheduled subsequently. Deciding the feasibility of smaller subproblems is expected to be considerable faster than deciding the overall scheduling problem. If necessary, the conflict analysis can be applied subsequently on the smaller subproblems which, in general, results in a faster deduction. In this paper, we distinguish the three following subproblems:

1. Schedule the computational tasks on each tile independently (TS).
2. Schedule the computational tasks on each tile including incoming and outgoing messages from the tile indepently (CS).
3. Schedule clusters of independent applications independently (AS).

In subproblem (AS), clusters of independent applications are defined such that no resource is shared between two clusters. Note that the subproblem (AS) may be equal to the problem of scheduling the complete system.

If a scheduling problem is proven to be unschedulable, a conflict analysis is applied to the respective problem. If one scheduling problem in a stage of the hierarchy is infeasible, the subproblems of the next stage are not considered. Instead, the results of all conflict analyses in that particular stage are used to restrict the search space of the answer set solver.

Conflict analysis. To derive a minimal reason why a feasible schedule cannot be derived in the subproblems (TS), we implemented the modified deletion filter presented in [5]. Compared to the deletion filter without modification [11], the algorithm does not iterate over all constraints of an infeasile problem. Instead, sets of constraints that influence the start time $\mathbf{s}_\lambda^r$ of a computational task or message λ are removed in one iteration. This approach often reduces the number of iterations in the deletion filter significantly. Iterating over all constraints of a scheduling problem does not result in a more expressive minimal reason because the indicator variables linking the answer set solver with the $\mathcal{T}$-solver do not reflect the expressiveness of the constraints in the scheduling problem. Conflict analysis of the the problems in (CS) and (AS) is based on the forward filter illustrated in [11]. Again, the algorithm was modified such that sets of constraints were removed per iteration in order to improve scalability.

Restriction of Search Space. Following the definition of hierarchical scheduling stage (TS), applying the modified deletion filter on a infeasible problem in (TS) translates to a set of computational tasks $\mathbf{T}$ that are responsible for the scheduling infeasibility. Similar to the work presented in [2], a constraint is added to the answer set solver that ensures that all the computational tasks in $\mathbf{T}$ are never bound to one tile in all subsequently computed bindings $\overline{\mathbf{B}}$, such that (6) holds.

$$\forall r \in \mathbf{R}_t, \forall \mathbf{B}^* \in \overline{\mathbf{B}} : \{(t,r) \mid t \in \mathbf{T}, (t,r) \in \mathbf{B}\} \not\subseteq \mathbf{B}^* \qquad (6)$$

In contrast to the conflict analysis in the (TS) stage, the result of the conflict analysis applied in the stages (CS) and (AS) may translate to a set $\mathbf{\Lambda}$ containing computational tasks and messages. Analog to [5], without breaking any symmetries, the search space is restricted by concrete binding of the computational tasks in $\mathbf{\Lambda}$ and the concrete routing of the messages in $\mathbf{\Lambda}$.

VI. Experimental Results

In this section, we present the experimental results showing the applicability of our proposed coordinated SMT-based synthesis approach. On basis of system specifications with increasing complexity, we compare the synthesis with and without a scheduling-aware refinement during the deviation of a B&R (cf. Fig. 2).

Experimental Setup. To decide the B&R of the applications, we employed the answer set solver `clingo`[1] [13]. With the exception of the conflict analysis using the forward filter, we employed the SMT-solver `yices`[2] [14] as $\mathcal{T}$-solver. The conflict analysis implementing the forward filter was realized using the SMT-solver `Z3` (version 4.3.1; [15]) in logic `QF_IDL`. The maximal time to derive one feasible implementation per synthesis run was set to $3{,}600s$. After this time, a still running synthesis attempt times out. In the coordinated approach, the time budget assigned to the answer set solver in order to refine the B&R was set to $10s$. Further, we set the maximum utilization of a tile to $U_{max} = 95$. All experiments were performed on a dual-processor Linux workstation implementing two quad-core 2.4GHz Intel Xeon E5620 with 48GB RAM. All synthesis runs utilized at most one CPU core and were carried out sequentially. To exclude noise effects, all reported results are the mean over three individual synthesis runs. The results of all three runs were always consistent, i.e, either all three individual runs provided a feasible implementation or all three runs timed out. Further, it should be noted that synthesis times of the coordinated approach for different runs of the same instance can be different in a limited but more noticeable range. This can be explained by the fact that the operating system timer jitters minimal around the 10s for the refinement. However, the minimal deviation in the timing may lead to an interrupt in different positions in the search space of the answer set solver. A subsequent restriction of the search space hence can lead to different starting points for a new search and possibly a different time until a new solution is derived.

Testcases. The NoC-based platforms of our testcases were regular NxN 2D-meshes with $N \in \{3,4,5\}$, resulting in a total number of N^2 tiles in the respective platform. The delays of messages on the resources of the platform's NoC were set to $\forall r \in \mathbf{R}_l : C_m^r = 5$ cycles and $\forall r \in \mathbf{R}_{rsu} : C_m^r = 7$ cycles. We generated 12 synthetic application set instances per platform. To generate a synthetic application set $\mathbf{A}$ of a testcase, randomized applications A_i were added incrementally to the application set until the overall computational utilization of the platform U_{sys} was greater than or equal to 40% of the maximum utilization:

$$U_{sys} = \sum_{t \in \mathbf{T}_i^t,\ A_i \in \mathbf{A}} \frac{C_t}{P_i} \geq (0.4 \cdot N^2). \qquad (7)$$

Each randomized application $A_i = (g_A^i, P_i, D_i)$ has a random period and deadline $D_i = P_i \in \{5ms, 10ms, 20ms, 40ms, 80ms\}$ and consists of a random number of tasks $|\mathbf{T}_i^t| \in \{3,4,5,6\}$. For each computational task, a random utilization $C_t/P_i \in \{3\%,4\%,5\%,6\%\}$ is chosen. The overall number of messages in an application is $|\mathbf{T}_i^t| - 1$. Exchange of messages between computational tasks in an application was specified such that the resulting application graph g_A^i was a task chain (cf. the application graph in Fig.3).

Table I shows the average number of applications, computational tasks and messages of the generated 12 instances in a testcase together with the respective standard deviation. Further, the average number of variables and constraints that represent the B&R-problem of an instance in the answer set solver is shown. The testcases denoted A-X summarize the synthesis runs without refinement whereas the testcases B-X synthesized the same instances as A-X but exploited the coordinate synthesis we propose in this paper. As can be seen, enabling the scheduling-aware refinement of a B&R solution results in noticeable larger problem instances for the answer set solver (cf. Section V). If we compare the testcase A-5 with the testcase B-5, the average number of variables and the average number of constraints of the 12 instances is more than doubled in a coordinated synthesis.

Results and Discussion. Table II shows the experimental results of the six synthesis testcases. We observed that the

[1]Version 4.3.0. Command-line arguments were `--opt-strategy=5` and `--configuration=frumpy`.

[2]Version 2.2.1. With command-line arguments `--logic=QF_IDL` and `--arith-solver=floyd-warshall`.

TABLE I. Overview of the testcases & instances

Testcase	Platform	Instances	Avg. Applications	Avg. Tasks	Avg. Messages	Synthesis Mode	Avg. Variables (B&R)	Avg. Constraints (B&R)
A-3	3x3	12	18 ± 1	80 ± 2	62 ± 2	w/o coordination	31,962 ± 987	84,229 ± 2,623
B-3	3x3	12	18 ± 1	80 ± 2	62 ± 2	w/ coordination	45,702 ± 1,706	128,340 ± 4,901
A-4	4x4	12	32 ± 2	142 ± 4	110 ± 3	w/o coordination	113,385 ± 2,635	301,929 ± 7,032
B-4	4x4	12	32 ± 2	142 ± 4	110 ± 3	w/ coordination	200,031 ± 10,568	571,251 ± 30,900
A-5	5x5	12	49 ± 1	222 ± 4	172 ± 4	w/o coordination	296,313 ± 6,299	793,546 ± 16,951
B-5	5x5	12	49 ± 1	222 ± 4	172 ± 4	w/ coordination	617,430 ± 17,704	1,781,432 ± 52,169

TABLE II. Synthesis Results

Testcase	Timeouts	Avg. Synthesis Time (Success) $[s]$	Avg. Couplings (Success / Timeout)	Avg. B&R (Success) $[s]$	Avg. Scheduling (Success) $[s]$
A-3	7 of 12	1,255.0 ± 1,211.6	8 ± 7 / 1 ± 1	0.5 ± 0.2	1,254.5 ± 1211.5
B-3	–	39.0 ± 44.4	3 ± 3 / –	38.7 ± 44.1	0.3 ± 0.3
A-4	8 of 12	552.2 ± 364.7	6 ± 2 / 2 ± 1	1.8 ± 0.2	550.4 ± 364.5
B-4	–	31.5 ± 26.8	2 ± 2 / –	31.3 ± 26.7	0.2 ± 0.1
A-5	11 of 12	742.7 ± 0.0	6 ± 0 / 4 ± 5	5.2 ± 0.0	737.5 ± 0.0
A-5	–	229.6 ± 126.9	8 ± 5 / –	228.6 ± 126.3	1.0 ± 0.6

synthesis approach without the refinement of B&R solutions is not able to synthesize over half of the instances in a testcase within the timelimit of $3{,}600s$. Particular in testcase A-5 only a single instance was synthesized. In contrast, the coordinated approach was always able to derive a feasible implementation within the timelimit.
Concerning the average synthesis time until a feasible implementation is found for all successfully synthesized instances, Table II indicates that the coordinated approach often results in lower overall synthesis times. Actually, 2 of the overall 10 successfully synthesized instances utilizing the synthesis without refinement show a lower average synthesis time compared to the equivalent time measured for these two instances using the coordinated approach. Specifically, this concerns one instance with 9 tiles ($6.7s \pm 0.0s$ vs. 162.6 ± 17.0) and one instance with 16 tiles ($11.3s \pm 0.0s$ vs. $73.5s \pm 16.4s$).
The last two columns of Table II show the average overall time spent for B&R and scheduling of a successful synthesis. As one can see, the time related with B&R in the synthesis is noticeably higher in the coordinated synthesis approach. On the one hand, this is due to the additional 10s for the refinement of solutions. On the other hand, the additional variables and constraints (cf. Table I) due to the formulation of the refinement objective (cf. Section V) increase the time until an initially valid B&R is derived. The overall average time in a synthesis to derive initial B&R solutions can be estimated by dividing the average overall B&R time with the average number of couplings and subtracting the 10s for refinement. For the testcases B-4, this estimated time is $5.7s$ which is 19 times larger than the equivalent time in A-4.
However, in the coordinated approach the effort to refine a B&R enables the significant reduced time to compute feasible schedules as shown in the last column of Table II. Thus, more complex systems can be synthesized. For synthesis runs that timed out, nearly $3{,}600s$ are spent in the scheduling part of the system synthesis, as illustrated in Fig. 1.

VII. Concluding Remarks

We proposed an SMT-based synthesis approach for periodic applications mapped onto time-triggered tile-based platforms where the logic solver is coordinated with the $\mathcal{T}$-solver. Thereby, the coordination is realized by an additional time budget assigned to the logic solver in order to refine the binding and routing. In contrast to a synthesis without this scheduling-aware refinement, we demonstrated that the $\mathcal{T}$-solver decided schedulability within a reasonable amount of time. Despite the additional time consuming refinement in the logic solver, our experiments show that the overall synthesis time can be considerably reduced and more complex systems can be synthesized.

References

[1] 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.

[2] 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.

[3] M. Lukasiewycz, M. Streubuhr, M. Glass, C. Haubelt, and J. Teich, "Combined System Synthesis and Communication Architecture Exploration for MPSoCs," in Proc. of DATE, 2009, pp. 472–477.

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

[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] P. Kumar, D. B. Chokshi, and L. Thiele, "A Satisfiability Approach to Speed Assignment for Distributed Real-Time Systems," in Proc. of DATE, 2013, pp. 749–754.

[7] N. Satish et al., "A Decomposition-based Constraint Optimization Approach for Statically Scheduling Task Graphs with Communication Delays to Multiprocessors," in Proc. of DATE, 2007, pp. 1–6.

[8] W. Liu et al., "Satisfiability Modulo Graph Theory for Task Mapping and Scheduling on Multiprocessor Systems," in IEEE Trans. on Parallel and Distributed Systems, vol. 22, no. 8, 2011, pp. 1382–1389.

[9] E. K. Jackson et al., "Components, Platforms and Possibilities: Towards Generic Automation for MDA," in Proc. of EMSOFT, 2010, pp. 39–48.

[10] B. Andres, M. Gebser, T. Schaub, C. Haubelt, F. Reimann, and M. Glaß, "Symbolic System Synthesis Using Answer Set Programming," in Proc. of LPNMR, 2013, pp. 79–91.

[11] M. Ostrowski and T. Schaub, "ASP modulo CSP: the clingcon system," Theory Pract. Log. Program., vol. 12, no. 4-5, 2012, pp. 485–503.

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

[13] M. Gebser et al., "*Clingo* = ASP + control: Preliminary report," in Technical Communications of ICLP, vol. 13, no. 4-5, 2014.

[14] B. Dutertre, "Yices 2.2," in Proc. of CAV, 2014, pp. 737–744.

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