Symbolic System Synthesis in the Presence of Stringent Real-Time Constraints*

Felix Reimann†, Martin Lukasiewycz‡, Michael Glass†, Christian Haubelt†, Jürgen Teich†

†University of Erlangen-Nuremberg, Germany
{felix.reimann,glass,haubelt,teich}@cs.fau.de

‡TU Munich, Germany
martin.lukasiewycz@rcs.ei.tum.de

Abstract— Stringent real-time constraints lead to complex search spaces containing only very few or even no valid implementations. Hence, while searching for a valid implementation a substantial amount of time is spent on timing analysis during system synthesis. This paper presents a novel system synthesis approach that efficiently prunes the search space in case real-time constraints are violated. For this purpose, the reason for a constraint violation is analyzed and a deduced encoding removes it permanently from the search space. Thus, the approach is capable of proving both the presence and absence of a correct implementation. The key benefit of the proposed approach stems from its integral support for real-time constraint checking. Its efficiency, however, results from the power of deduction techniques of state-of-the-art Boolean Satisfiability (SAT) solvers. Using a case study from the automotive domain, experiments show that the proposed system synthesis approach is able to find valid implementations where former approaches fail. Moreover, it is up to two orders of magnitude faster compared to a state-of-the-art approach.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems

General Terms

Design, Algorithms

1. INTRODUCTION

The complexity of nowadays and future embedded systems is growing tremendously. Coping with a steadily increasing number of applications and resources, heterogeneity, distributed implementations, and stringent constraints arising from safety, reliability, and control aspects has become the key challenge for automatic design approaches at the *Electronic System Level* (ESL). An essential task during ESL design is *system synthesis* that determines a system implementation by: (1) *allocating* processors, hardware accelerators, network controllers, and memories, (2) *binding* tasks onto computational resources, (3) *routing* messages on the communication resources, and (4) *scheduling* computation tasks and communication tasks. This work proposes a novel symbolic approach that targets system synthesis in the presence of stringent real-time constraints.

The constraints that have to be met by the system implementation can be coarsely categorized into functional and nonfunctional constraints. *Functional constraints* assure a *feasible* composition of the system. Examples are: tasks can only be executed on allocated resources; data-dependent tasks must be able to communicate; or each task has to be assigned a unique priority. In platform-based design, functional constraints can be formulated as selection and assignment problems. In recent years, approaches have been developed that are capable of respecting such constraints by construction, e. g., by using formal techniques like SAT or *Integer Linear Programming* (ILP) solving. Besides checking functional constraints, these formal techniques can also be used for checking *linear* nonfunctional constraints [8]. However, the characteristic of typical *nonfunctional constraints* such as real-time constraints prohibits a linearization. Thus, such nonfunctional constraints cannot be respected by construction and make a special treatment during system synthesis mandatory. Implementations that do not fulfill given nonfunctional constraints are said to be *invalid*. Implementations satisfying all functional and nonfunctional constraints are considered to be *correct*.

A recently published system synthesis approach supports checking of generic nonlinear nonfunctional constraints using *Satisfiability Modulo Theories* (SMT) solving [12]: Functional constraints are encoded by BOOLEAN formulas and SAT solving is used to find feasible solutions. Each found implementation is evaluated[1] and, in case of invalidity, forbidden by adding a corresponding clause to the BOOLEAN formula. As a result, invalid implementations are iteratively pruned from the search space. Thus, this approach either returns a correct implementation or it proves that the search space only consists of incorrect implementations. If only monotonous constraints are considered, the efficiency can be further improved by checking nonfunctional constraints on partial implementations.

When trying to apply the approach presented in [12] in the presence of stringent real-time constraints, a major drawback appears: In case an implementation or a partial implementation is invalid, exactly *this* (partial) implementation including allocation, binding, and the scheduling of all tasks and messages is forbidden. A closer look, however, reveals that typically one or several critical paths are responsible for the violation of given real-time constraints. Even more

*Supported in part by the German Ministry for Research and Education (BMBF Grant SEIS 01BV0910).

DAC'11, June 5-10, 2011, San Diego, California, USA

[1] For the evaluation, arbitrary analysis techniques based on formal analysis or simulation are applicable.

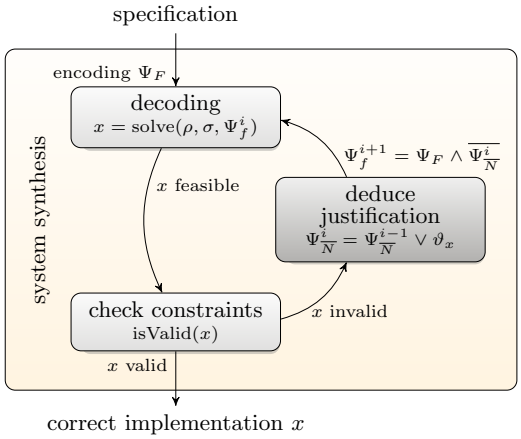


Figure 1: Proposed system synthesis approach: The justifications are those design decisions that are responsible for the violation of given real-time constraints.

important, many binding, routing, as well as schedule design decisions do not influence the critical path at all; they are *ineffective* to the critical path. As a result, the former approach has to investigate all implementations that potentially have the same critical path since it is not aware of the ineffectiveness of several decisions.

Contributions. A novel symbolic system synthesis approach is proposed that uses a background theory to resolve the reason for the invalidity of an implementation regarding real-time constraints. The resulting *justification* [2] corresponds to those design decisions which are effective to the critical paths of an invalid implementation. This justification, propagated to the underlying SAT solver, now enables for an efficient search for correct implementations or a proof of the absence of them is provided. The overall approach is depicted in Fig. 1. Compared to the former, generic approach published in [12], significantly larger parts of the design space can be pruned using fewer of the typically expensive timing evaluations. To enable the *deduction* of the justification from scheduling decisions, the work at hand introduces an efficient encoding of unique priorities by means of a relative order. By this, nonlinear nonfunctional constraint checking has become an integral part of the proposed system synthesis approach. Experimental results give evidence that the proposed approach is superior to previously published approaches. In particular, it finds correct implementations where former approaches that use fixed schedules fail. Moreover, it is up to two orders of magnitude faster in proving the absence of a correct implementation compared to the generic approach.

2. RELATED WORK

Today, *platform-based design* [6] is a concept that is followed by the majority of available ESL design tools, see for example [3, 7, 11]. It enables to formulate the tasks of system synthesis, i. e., allocation, binding, and scheduling, as a combinatorial decision problem. In order to take functional constraints in the combinatorial decision problem into account, symbolic approaches have become the first choice in recent years. Known approaches employ ILP [10], SAT solving [5], or Binary Decision Diagrams (BDDs) [9]. Applied in various domains, these approaches efficiently cope with functional constraints. An extension of these techniques is proposed in [8] where the authors consider linear or linearizable nonfunctional constraints and formulate the system synthesis problem as a 0–1 ILP. However, this linearization is only reasonably applicable for particular nonfunctional constraints like the load on a bus or processor and cannot be applied to real-time constraints as covered in the work at hand.

In [12], we propose a first SMT-based approach to integrate nonfunctional constraint checking in the SAT-based allocation and binding computation. This generic approach is sufficient to prune invalid implementations, but does not cover the aspect of analyzing the reason for invalidity. Although real-time analysis is used as a case-study, no encoding scheme for scheduling information is presented and, thus, the experimental results are based on fixed predefined scheduling policies and priorities.

To the best of our knowledge, there exists no other approach that seamlessly combines a formal encoding for allocation, binding, routing, scheduling, and linearizable constraints with the ability to efficiently cope with nonlinear constraints. In the following, a novel encoding scheme to integrate scheduling in the formal model is presented. Afterwards, the system synthesis approach is introduced that analyzes invalid implementations to extract only those design decision that are responsible for the constraint violation, i. e., the *justification*. Deducing the justification only, the methodology can focus on the reasons for constraint violations while known approaches typically decay to an exhaustive search algorithm by taking design decisions into account that are ineffective to the violation.

3. PROBLEM FORMULATION

A graph-based specification of a system synthesis problem is given by $g_S(g_T, g_A, M, \mathcal{C}_F, \mathcal{C}_N)$. A set of applications are represented as a bipartite *application graph* $g_T = (T \cup C, E_T)$. Each vertex $t \in T$ models a computational task and each vertex $c \in C$ a communication task. An edge $e \in E_T \subseteq (T \times C) \cup (C \times T)$ represents control or data dependencies between computational and communication tasks or vice versa. The set of available resources is given by the *architecture graph* $g_A = (R, E_A)$. Vertices $r \in R$ model resources like processors, memories, or buses. Edges $E_A \subseteq R \times R$ model connections between these. Beside the application graph g_T and the architecture graph g_A, a set of *mapping edges* $M \subseteq T \times R$ is given. Each mapping edge $m = (t, r) \in M$ indicates a possible mapping of a computational task t onto a resource r.

System synthesis computes an *allocation* $A \subseteq R$, a *binding* $B \subseteq M$, a *routing* of each communication task $c \in C$ on a tree of resources $W_c \subseteq R$, and a schedule S. An allocation A is a subset of resources used in the implementation. A binding B is a subset of mappings that assign computational tasks to resources. A routing W_c is a subset of resources over which the communication is routed. The schedule S can be static (predefined start times for the tasks) or dynamic (assigning priorities or deadlines to tasks).

An *implementation* $x = (A, B, W, S) \in X$ is one possible solution in the search space $X = 2^R \times 2^M \times 2^{C \times R} \times 2^S$. Any correct implementation has to obey *functional constraints* $\mathcal{C}_F$ and *nonfunctional constraints* $\mathcal{C}_N$. Thus, system synthesis can be formally defined as:

Problem Find an implementation $x \in X_f \subseteq X$ with $X_f = X_F \cap X_N$ being the set of *correct* implementations as an intersection of the feasible implementations X_F that fulfill the given functional constraints $\mathcal{C}_F$ and the valid implementations X_N that satisfy the given nonfunctional constraints $\mathcal{C}_N$.

Basic functional constraints $\mathcal{C}_F$ impose that:

1. each resource that shall execute a computational task is allocated, i. e., $\forall (t,r) \in B : r \in A$;
2. each computational task is bound exactly once, i. e., $\forall t \in T : |\{m \in B \mid m = (t,r)\}| = 1$;
3. each message can be routed on the allocated architecture to fulfill the control or data dependencies of the communicating tasks; and
4. a unique priority is assigned to each computational and communication task of the application g_T.

Any quadruple of allocation, binding, routing, and scheduling fulfilling these functional constraints is called *feasible*. Note that each feasible implementation $x \in X_F$ and, hence, each correct implementation $x \in X_f$ is based on such a feasible allocation, binding, routing, and scheduling.

4. PROBLEM ENCODING

This section presents the unified encoding of allocation, binding, routing, and schedule decisions with respect to functional constraints as a 0–1 ILP. For the sake of completeness, the 0–1 ILP that takes into account allocation, binding, and routing is introduced first. Afterwards, the encoding of relative priorities is presented such that the consideration of priority-based scheduling strategies is enabled.

For the encoding, a *characteristic function* $\Psi : 2^{\Theta} \to \{0,1\}$ of X_F is defined that encodes all feasible implementations:

$$\begin{aligned}\Theta \;=\; & \{\mathbf{r} \mid \forall r \in R : \mathbf{r} \in \{0,1\}\} \;\cup \\ & \{\mathbf{m} \mid \forall m = (t,r) \in M : \mathbf{m} \in \{0,1\}\} \;\cup \\ & \{\mathbf{c}_r \mid \forall c \in C, r \in R : \mathbf{c}_r \in \{0,1\}\} \;\cup \\ & \{\mathbf{c}_{r\tau} \mid \forall c \in C, r \in R, \tau \in \mathbb{T} : \mathbf{c}_{r\tau} \in \{0,1\}\}\end{aligned}$$

Here, the activation of a resource $r \in R$ is encoded by the truth assignment $\mathbf{r} := 1$ and the selection of a mapping possibility $m = (t,r) \in M$ is encoded by the truth assignment $\mathbf{m} := 1$. If a communication task c is routed over a resource r it is encoded by $\mathbf{c}_r := 1$ and if this happens at the time step $\tau \in \mathbb{T}$, $\mathbf{c}_{r\tau} := 1$.

4.1 Allocation, Binding, and Routing

The characteristic function Ψ_F that takes into account allocation, binding, and routing is defined as follows:
$\forall t \in T$:

$$\textstyle\sum_{m=(t,r)\in M} \mathbf{m} = 1 \tag{1a}$$

$\forall c \in C$:

$$\textstyle\sum_{r\in R} \mathbf{c}_{r0} = 1 \tag{1b}$$

$\forall c \in C, r \in R, (t,c) \in E_T, m = (t,r) \in M$:

$$\mathbf{m} - \mathbf{c}_{r0} = 0 \tag{1c}$$

$\forall c \in C, r \in R, (c,t) \in E_T, m = (t,r) \in M$:

$$\mathbf{c}_r - \mathbf{m} \geq 0 \tag{1d}$$

$\forall c \in C, r \in R$:

$$\textstyle\sum_{\tau\in\mathbb{T}} \mathbf{c}_{r\tau} \leq 1 \tag{1e}$$

$$\textstyle\sum_{\tau\in\mathbb{T}} \mathbf{c}_{r\tau} - \mathbf{c}_r \geq 0 \tag{1f}$$

$\forall c \in C, r \in R, \tau \in \mathbb{T}$:

$$\mathbf{c}_r - \mathbf{c}_{r\tau} \geq 0 \tag{1g}$$

$\forall c \in C, r \in R, \tau = \{1, \ldots, |\mathbb{T}|\}$:

$$\left(\textstyle\sum_{\tilde{r}\in R,(\tilde{r},r)\in E_A} \mathbf{c}_{\tilde{r}\tau}\right) - \mathbf{c}_{r(\tau+1)} \geq 0 \tag{1h}$$

$\forall m = (t,r) \in M$:

$$\mathbf{r} - \mathbf{m} \geq 0 \tag{1i}$$

$\forall c \in C, r \in R$:

$$\mathbf{r} - \mathbf{c}_r \geq 0 \tag{1j}$$

$\forall r \in R$:

$$-\mathbf{r} + \textstyle\sum_{c\in C, r\in R} \mathbf{c}_r + \sum_{m=(t,r)\in M} \mathbf{m} \geq 0 \tag{1k}$$

Eq. (1a) ensures that each computational task is bound exactly once. Eq. (1b) and Eq. (1c) imply that each communication task has exactly one root that equals the used resource of the sending task. Analogously, for each receiving task the predecessor communication tasks have to be routed to the corresponding resources as stated in Eq. (1d). Eq. (1e) prohibits cycles as it ensures that a communication task can pass a resource at most once. A communication task has to be existent in one communication step on a resource in order to be correctly routed on this resource as implied by Eq. (1f) and Eq. (1g). Eq. (1h) states that a routing is only possible between adjacent resources. Eq. (1i) and Eq. (1j) imply that a computational or communication task, respectively, is bound or routed on an allocated resource only. On the other hand, Eq. (1k) states that a resource is only allocated if at least one computational or communication task is bound or routed on this resource. Any consistent truth assignment such that the characteristic function Ψ_F evaluates to 1 represents a feasible allocation, binding, and routing with respect to functional constraints.

4.2 Priority Assignment

In order to support various scheduling schemes, a symbolic encoding for task priorities is proposed. Binary variables $\mathbf{p}_{ij} \in \{0,1\}$ are introduced that indicate whether a task i has a higher priority than a task j ($\mathbf{p}_{ij} = 1$) or not ($\mathbf{p}_{ij} = 0$). The encoding of priorities requires to add the following constraints to the characteristic function Ψ_F:
$\forall i,j,k \in C, i \neq j \neq k$:

$$\mathbf{p}_{ij} + \mathbf{p}_{jk} + \overline{\mathbf{p}_{ik}} \leq 2 \tag{2a}$$

$$\overline{\mathbf{p}_{ij}} + \overline{\mathbf{p}_{jk}} + \mathbf{p}_{ik} \leq 2 \tag{2b}$$

$\forall r \in R, (i,r), (j,r), (k,r) \in M, i \neq j \neq k$:

$$\mathbf{p}_{ij} + \mathbf{p}_{jk} + \overline{\mathbf{p}_{ik}} \leq 2 \tag{2c}$$

$$\overline{\mathbf{p}_{ij}} + \overline{\mathbf{p}_{jk}} + \mathbf{p}_{ik} \leq 2 \tag{2d}$$

$\forall r \in R, (i,r), (j,r) \in M, i \neq j$:

$$\mathbf{p}_{ij} + \overline{\mathbf{p}_{ji}} = 1 \tag{2e}$$

$\forall i,j \in C, i \neq j$:

$$\mathbf{p}_{ij} + \overline{\mathbf{p}_{ji}} = 1 \tag{2f}$$

The Eqs. (2a)–(2d) ensure that the priorities are transitive, i. e., if the priority of task i is greater than the priority of task j and the priority of task j is greater than the priority of task k, then the priority of task i must also be greater than the priority of task k, $\mathbf{p}_{ik} \stackrel{!}{=} 1$. This can be either encoded globally, as done for the communication tasks in Eqs. (2a) and (2b), or on a per resource basis, as done for the computational tasks in Eqs. (2c) and (2d). Eqs. (2e) and (2f) ensure that if task i has a higher priority than task j ($\mathbf{p}_{ij} := 1$) then task j has lower priority than task i, $\mathbf{p}_{ji} \stackrel{!}{=} 0$. The same holds for the communication priority variables. Since reflexivity and antisymmetry are satisfied by default, the transitivity constraints ensure a partial order on the priorities. Thus, a unique priority assignment is enabled.

4.3 Symbolic System Synthesis

Given the 0–1 ILP formulation of the characteristic function Ψ_F, the problem targeted in this work is given as:

> **Problem** Find a truth assignment ϑ such that $\Psi_f = 1$ with $\Psi_f = \Psi_F \wedge \Psi_N$ being the characteristic function of the set of correct implementations encoded as the conjunction of the characteristic functions Ψ_F and Ψ_N.

Here it is assumed that the set of all valid implementations can be represented by a characteristic function Ψ_N. In the following section, it is shown how Ψ_N can efficiently be approximated during system synthesis. To derive a correct implementation, a SAT solver is used to find a satisfying truth assignment to Ψ_f, i. e., $\exists \vartheta : \Psi_f$. Conceptually, a SAT solver traverses a *decision tree* where each vertex represents a variable and the outgoing edges an assignment of 0 or 1. A systematic search on this tree is performed such that a path from the root to a leaf equals a feasible truth assignment ϑ. The search is performed as a *backtracking* such that if no leaf exists, the function Ψ is identified as *unsatisfiable*.

If a satisfying truth assignment ϑ is found, the allocation A, binding B, routing W, and priority assignment S as given in Eq. (3a)–(3d) are deduced in order to determine the implementation x:

$$A = \{r \in R \mid \mathbf{r} = 1\} \tag{3a}$$

$$B = \{m \in M \mid \mathbf{m} = 1\} \tag{3b}$$

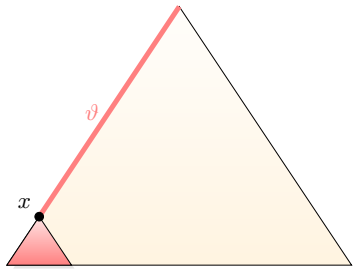
$$W = \{(c, r) \in C \times R \mid \mathbf{c}_r = 1\} \tag{3c}$$

$$S \;:\; T \to \mathbb{N} \text{ with } S(t) = \sum_{t,\widetilde{t} \in T, \widetilde{t} \neq t, m_{tr} \in B} \mathbf{p}_{t\widetilde{t}} \cdot \widetilde{\mathbf{m}}_{\widetilde{t}r}, \tag{3d}$$

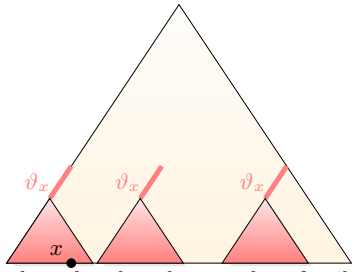
$$C \to \mathbb{N} \text{ with } S(c) = \sum_{t,\widetilde{t} \in T, \widetilde{t} \neq t} \mathbf{p}_{t\widetilde{t}} \tag{3e}$$

5. DEDUCING JUSTIFICATIONS

Let x be a feasible implementation according to the characteristic function Ψ_F. If the timing analysis of x reveals one or more deadline violations, i. e., $x \notin \Psi_N$, x is also incorrect and $x \notin \Psi_f$. Avoiding that this implementation is considered again during system synthesis requires to forbid the corresponding truth assignment ϑ. In contrast to previous work, not the complete truth assignment ϑ is forbidden, but solely the *justification*, i. e., those truth assignments that induce the deadline violations, are excluded. The arising advantage is depicted in Fig. 2: (a) excluding the complete truth assignment ϑ only affects exactly one implementation or, in case of *online integration* [2] termed *early learning*



(a) In [12], the decision process is interrupted at specific barriers resulting in the truth assignment ϑ which corresponds to a partial implementation x. If this partial implementation is invalid, only the assignments in the subsequent part of the decision tree can be pruned from the search space.



(b) In the work at hand, only complete feasible implementations are evaluated. The justification ϑ_x of the implementation x affects several parts of the decision tree and, thus, enables a significant pruning of the search space.

Figure 2: The decision tree resulting from the characteristic function Ψ_F. The black dot represents the found feasible implementation x that is analyzed and found to be invalid.

in [12], an area below the invalid partial implementation in the decision tree; (b) excluding the justification ϑ_x affects all areas in the decision tree that include these truth assignments. Thus, analyzing an invalid implementation and deducing the justification allows to efficiently prune large parts of the search space. In the following, a brief introduction of the generic SMT-based system synthesis approach by [12] is given. Afterwards, the proposed approach to analyze real-time constraint violations is presented in order to extract the justification.

5.1 SMT-based System Synthesis

The work at hand employs the *lazy offline* [2] SMT-based system synthesis proposed as *simple learning* in [12]. It relies on the following observations: The set of correct implementations is given as $\Psi_f = \Psi_F \wedge \Psi_N$. However, there is no closed form of Ψ_N available, given the constraints that determine this set are nonfunctional and nonlinear. An alternative way to derive Ψ_f is based on the set of all implementations that are invalid $\Psi_{\overline{N}}$. Given this, the set of correct implementations is derived as $\Psi_f = \Psi_F \wedge \overline{\Psi_{\overline{N}}}$. SMT-based system synthesis uses this reformulation and aims to iteratively construct the set of invalid implementations $\Psi_{\overline{N}}$ by evaluating feasible implementations and adding invalid implementations to the set.

Let $\Psi_f^{(i)}$ describe the iteratively constructed set of correct implementations in the i-th iteration, the SMT-based system synthesis starts at iteration 0 with $\Psi_{\overline{N}}^{(0)} = 0$:

$$\Psi_f^{(1)} = \Psi_F \wedge \overline{\Psi_{\overline{N}}^{(0)}} = \Psi_F$$

In case x derived in step i is invalid, the respective encoding ϑ of x is combined with $\Psi_N^{(i)}$ using disjunction, see also Fig. 1:

$$\Psi_f^{(i+1)} = \Psi_F \wedge \overline{(\Psi_N^{(i)} \vee \vartheta)} \tag{4}$$

As a result, the approach is capable of approximating Ψ_f over time (iterations). In case Ψ_f decays to 0, i. e., the SAT solver detects a contradiction, the absence of any correct implementation is proven.

5.2 Deduction of the Justification

In the following, the deduction of the justification ϑ_x and, therefore, the reason for a deadline violation is presented. The justification is pruned from the search space according to Eq. (4). By deducing a justification which consists only of a subset of variables, i. e. $|\vartheta_x| \leq |\vartheta|$, according to the concrete reason for invalidity, this knowledge is directly incorporated in $\Psi_N^{(i)}$.

Let t be the task for which an employed timing analysis reports a deadline violation. Then $V_t \subseteq T \cup C$ is a set of tasks and communications which are on the critical path set of t, i. e., all predecessors of t in the application g_T. It follows: $V_t = Pre^n(t)$ with $Pre(t) = \{\tilde{t} | (\tilde{t}, t) \in E_T\}$ and $Pre^n(t) = Pre^{n-1}(t)$ as the fix point of iterative predecessor determination. Thus, the justification or the reason for the deadline violation at t is given by the combined time consumption of the mappings of the tasks and the routings of the communications in V_t. Consider Fig. 3: task t_2 violates its deadline and, thus, t_2 and all preceding vertices of the application result in the corresponding invalid set $V_{t_2} = \{t_1, c_1, c_2, t_2\}$. If t_3 also misses its deadline, V_{t_3} can be ignored as it is a superset of V_{t_2} and, thus, is already included in the justification of t_2. The second deadline violation at t_{10} results in another invalid set $V_{t_{10}} = \{t_8, c_7, t_9, c_8, t_{10}\}$. Note that t_6 affects the critical path since it is executed on the same resource as t_9 and of higher priority. However, t_6 is not part of the critical set but respected during the construction of the justification as introduced in the following.

The justification ϑ_x for the deadline violation at task t is given by

$$\vartheta_x = \bigwedge_{i \in V_t} v(i), \tag{5}$$

where $v(i)$ computes the justification of task i on one specific resource r according to the time dependency of the task or communication i on this resource. The timing dependency of a task or communication depends on the scheduler of the resource and according to this possibly on other tasks or communications which are also mapped or routed on this resource. Therefore, in the following, the dependency function v is defined according to the needs of different standard schedulers.

5.2.1 Constant delay

If a task t of an activated mapping $m = (t, r) \in B$ is always delayed by a constant amount of time in the worst case, the mapping variable is directly added to the path constraint:

$$v(t) = \mathbf{m} \tag{6}$$

The same holds if a communication c is delayed on a resource r for a constant amount of time:

$$v(c) = \mathbf{c}_r \tag{7}$$

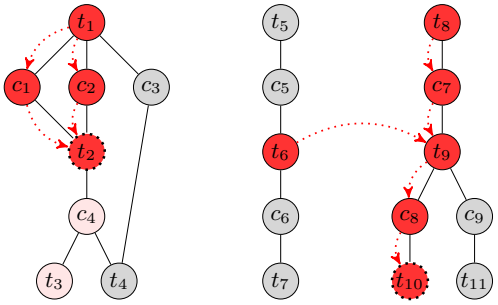


Figure 3: The application graph of an invalid implementation: deadline violations occur at task t_2 and t_{10}. Given the particular allocation, binding, routing, and scheduling, the dark nodes and dotted edges depict the design decisions that influence the critical path. Those are deduced as two justifications, one covering t_2 and one covering t_{10}, respectively. Other design decisions are ineffective to the critical paths.

In both cases it is not necessary to consider other tasks or communications that are mapped or routed on the same resource.

5.2.2 Preemptive fixed priority scheduling

Using static priority preemptive scheduling, the delay of a task t depends on (a) its execution time on the target resource and (b) all the tasks that are bound to the same resource and have a higher priority, see [1]. Thus, according to the truth assignment ϑ, let

$$\Theta_{tr} = \{\mathbf{m}_{\tilde{t}r} | \tilde{t} \in T, (\tilde{t}, r) \in B, \vartheta[\mathbf{p}_{\tilde{t}t}] = 1\} \tag{8}$$

be the mapping variable set of all tasks $\tilde{t}$ which are mapped to the same resource as task t and have a higher priority assigned. Correspondingly,

$$\Theta_{tr}^P = \{\mathbf{p}_{\tilde{t}t} | (t, r), (\tilde{t}, r) \in B, \vartheta[\mathbf{p}_{\tilde{t}t}] = 1\} \tag{9}$$

is the set of priority variables for the tasks with higher priority which are mapped to the same resource r as task t. It follows:

$$v(t) = \mathbf{m} \wedge \bigwedge_{\widetilde{\mathbf{m}} \in \Theta_{tr}} \widetilde{\mathbf{m}} \wedge \bigwedge_{\mathbf{p} \in \Theta_{tr}^P} \mathbf{p} \tag{10}$$

Thus, a constraint violation might not only be caused by the binding of the task itself, but also by all higher priority tasks on the same resource.

5.2.3 Non-preemptive fixed priority scheduling

For non-preemptive scheduling, the task t might be delayed by all tasks with higher priorities and at most one task with lower priority. To consider the worst case, the task t_{low} with $m_{\text{low}} = (t_{\text{low}}, r) \in B$ is selected such that it has a lower priority than task t and the longest execution time of all tasks with lower priority, see [4]. Therefore, in combination with Eq. (8) and Eq. (9), the function v is defined as follows:

$$v(t) = \mathbf{m} \wedge \bigwedge_{\widetilde{\mathbf{m}} \in \Theta_{tr}} \widetilde{\mathbf{m}} \wedge \bigwedge_{\mathbf{p} \in \Theta_{tr}^P} \mathbf{p} \wedge \mathbf{m}_{\text{low}} \tag{11}$$

Correspondingly, the function v is defined for each communication c. Here, event-triggered buses, e. g., the CAN bus, are modeled with non-preemptive fixed priority scheduling while time-triggered buses use a constant delay model.

6. EXPERIMENTAL RESULTS

In the following, the efficiency of the presented approach is investigated. Compared is: the SMT-based approach presented in [12] termed OFFLINE that takes into account a predefined scheduling only; an extended version of the SMT approach termed ONLINE that uses the schedule encoding proposed in the work at hand; and the novel deduction of justifications termed RTSMT. The used medium complex case study is an automotive subnet consisting of 15 computational resources, 2 CAN buses, and 1 FlexRay bus onto which 3 applications consisting of 27 tasks and 24 messages need to be mapped. The experiments are carried out on an Intel Pentium 4 3.00 GHz Dual Core machine with 1.5 GB RAM. The timing analysis is implemented using *Modular Performance Analysis* (MPA) in combination with *Real-Time Calculus* (RTC), see [1].

At first, OFFLINE is compared to RTSMT. In [12], it is proven after ≈ 255 minutes that there exists no correct implementation for this case study, given the assumed designer-specified schedule. In contrast, RTSMT is able to find a correct implementation in only 4 minutes. In fact, there exist 12 correct implementations when different schedules are considered as introduced in the work at hand. This highlights the importance of a unified allocation, binding, routing, and scheduling encoding.

Given the unfair comparison of OFFLINE and RTSMT with respect to scheduling, the extended approach ONLINE is used for the following investigations of the runtime of the generic SMT-based system synthesis and the deduction of justifications as proposed in this work. To take into account *incorrect* specifications, i. e., there exists not a single correct implementation, the deadlines given for the medium case study have been decreased by 5 % and 10 % such that three specifications MEDIUM, MEDIUM5, and MEDIUM10 are available. To highlight the scalability of the approach, a LARGE case study is created by adding another 4 artificial applications to the case study, more than doubling the number of tasks and messages. The results for the average runtime based on 10 test runs are depicted in Tab. 1. RTSMT shows a significant speedup compared to the generic SMT-based system synthesis that ranges from a factor of 10× up to more than 160×. This gives evidence that the proposed deduction allows for a more efficient pruning of the search space in combination with fewer costly timing evaluations needed. Especially targeting cases where a given platform is nearly maxed-out, low runtimes and, particularly, a provable presence or absence of correct implementations is a key enabler for a designer-interactive development of such systems.

Table 1: Comparison of the average runtime of the known SMT approach enhanced by the proposed schedule encoding and the novel approach based on justification deduction. All times are given in minutes.

test case	incorrect	ONLINE	RTSMT	speedup
MEDIUM		190.7	4.2	45.4×
MEDIUM5	✓	63.3	5.8	10.9×
MEDIUM10	✓	672.7	4.2	160.2×
LARGE	✓	30.0	0.7	42.8×

7. CONCLUSION

This work tackles the problem of system synthesis in the presence of stringent timing constraints. A novel symbolic system synthesis approach is proposed that resolves the reason for the invalidity of an implementation regarding real-time constraints. The new approach analyzes which design decisions are effective to the critical paths that violate real-time constraints, i. e., the justification. Deducing solely the justification, significantly larger parts of the design space can be pruned using fewer of the typically expensive timing analysis. Thus, the underlying SAT solver allows for an efficient search for correct implementations or delivers a proof of the absence of them. Given the used scheduling policies have a huge impact on the timing properties, this work introduces an efficient unified encoding of allocation, binding, routing, and scheduling. Experimental results have shown that: (a) taking scheduling into account enables finding correct implementations where a former SMT-based approach proved the absence of correct implementations based on a designer-predefined schedule; (b) deducing justifications leads to a significant speedup that ranges up to two orders of magnitude compared to an state-of-the-art SMT-based approach. By this, nonlinear nonfunctional constraint checking and, particularly, the provability of presence or absence of correct system implementations has become an integral part of the proposed system synthesis approach.

8. REFERENCES

[1] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *Proc. of DATE'03*, pages 190–195, 2003.

[2] L. de Moura and N. Bjørner. Satisfiability modulo theories: An appetizer. In *Formal Methods: Foundations and Applications*, pages 23–36. 2009.

[3] R. Dömer, A. Gerstlauer, J. Peng, D. Shin, L. Cai, H. Yu, S. Abdi, and D. Gajski. System-on-Chip Environment: A SpecC-based Framework for Heterogeneous MPSoC Design. *EURASIP J. on Embedded Systems*, 2008(647953):13, 2008.

[4] W. Haid and L. Thiele. Complex task activation schemes in system level performance analysis. In *Proceedings of CODES+ISSS'07*, pages 173–178. ACM, 2007.

[5] C. Haubelt, J. Teich, R. Feldmann, and B. Monien. SAT-based techniques in system design. In *Proc. of DAC '03*, pages 1168–1169, 2003.

[6] K. Keutzer, A. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli. System-Level Design: Orthogonalization of Concerns and Platform-Based Design. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 19(12):1523–1543, 2000.

[7] S. Kwon, Y. Kim, W.-C. Jeun, S. Ha, and Y. Paek. A Retargetable Parallel Programming Framework for MPSoC. *ACM Trans. Design Automation of Electronic Systems*, 13(3), 2008.

[8] M. Lukasiewycz, M. Glaß, C. Haubelt, J. Teich, R. Regler, and B. Lang. Concurrent topology and routing optimization in automotive network integration. In *Proceedings of DAC'08*, pages 626–629, June 2008.

[9] S. Neema. *System Level Synthesis of Adaptive Computing Systems*. PhD thesis, Vanderbilt University, Nashville, Tennessee, May 2001.

[10] R. Niemann and P. Marwedel. An algorithm for hardware/software partitioning using mixed Integer Linear Programming. *Design Automation for Embedded Systems*, 2(2):165–193, 1997.

[11] H. Nikolov, M. Thompson, T. Stefanov, A. D. Pimentel, S. Polstra, R. Bose, C. Zissulescu, and E. F. Deprettere. Daedalus: Toward Composable Multimedia MP-SoC Design. In *Proc. of DAC '08*, pages 574–579, 2008.

[12] 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'10*, pages 135–144, Oct. 2010.