Research Article

Answer Set versus Integer Linear Programming for Automatic Synthesis of Multiprocessor Systems from Real-Time Parallel Programs

Harold Ishebabi,¹ Philipp Mahr,¹ Christophe Bobda,¹ Martin Gebser,² and Torsten Schaub²

¹ Professorship for Computer Engineering, Institute for Computer Science, University of Potsdam, August-Bebel-Strasse 89, D-14482 Potsdam, Germany

² Professorship for Knowledge Processing and Information Systems, Institute for Computer Science, University of Potsdam, August-Bebel-Strasse 89, D-14482 Potsdam, Germany

Correspondence should be addressed to Harold Ishebabi, ishebabi@cs.uni-potsdam.de

Received 12 March 2009; Accepted 24 August 2009

Recommended by Lionel Torres

An automated design approach for multiprocessor systems on FPGAs is presented which customizes architectures for parallel programs by simultaneously solving the problems of task mapping, resource allocation, and scheduling. The latter considers effects of fixed-priority preemptive scheduling in order to guarantee real-time requirements, hence covering a broad spectrum of embedded applications. Being inherently a combinatorial optimization problem, the design space is modeled using linear equations that capture high-level design parameters. A comparison of two methods for solving resulting problem instances is then given. The intent is to study how well recent advances in propositional satisfiability (SAT) and thus Answer Set Programming (ASP) can be exploited to automate the design of flexible multiprocessor systems. Integer Linear Programming (ILP) is taken as a baseline, where architectures for IEEE 802.11 g and WCDMA baseband signal processing are synthesized. ASP-based synthesis used a few seconds in the solver, faster by three orders of magnitude compared to ILP-based synthesis, thereby showing a great potential for solving difficult instances of the automated synthesis problem.

Copyright © 2009 Harold Ishebabi et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

1. Introduction

In order to build flexible systems that can be adapted to applications, researchers have explored FPGA-based multiprocessor systems in an attempt to exploit both high-level parallelism in applications and the flexibility of reconfigurable devices, targeting both single [1, 2] and multiFPGA platforms [3, 4].

The process of implementing such systems is a very complex undertaking, consisting of phases such as the design of constituent IP-blocks (e.g., processors, memories, and buses), task mapping and architecture determination (high-level synthesis), low-level system integration, and finally, FPGA synthesis and placement and routing. The focus of this

work is on task-mapping and high-level synthesis, and builds up on a design platform that targets system integration and synthesis [1].

An automated architecture synthesis methodology based on combinatorial optimization is used to simplify the design process. This methodology addresses the problem of determining application-specific optimum system architectures as well as mapping and scheduling corresponding parallel programs.

Often, an optimum solution requires the sharing of processor resources between tasks, necessitating the use of a task scheduler, whose impact on the overall solution must be considered. In cases where cooperative schedules suffice, that is, for applications without strict timing requirements,

the resulting analysis during optimization is straightforward. This is because the overhead is easy to compute. In that case, the overall optimization objective is simply to minimize the overall execution time of the application, the makespan, or alternately, to explore area-throughput-power tradeoff.

However, applications which impose deadline guarantees for periodic tasks may require preemptive schedulers. In such situations, one must determine how often task switching actually takes place, depending on task priorities and the schedule. The reason is that the schedule has a direct impact on the overall execution time, and hence on the optimum task mapping and resource allocation. Moreover, the optimization process must take into account schedulability constraints, because some task mappings may not guarantee deadlines.

The method presented in this paper considers the effect of fixed-priority scheduling during architecture synthesis. This covers a broad spectrum of embedded applications, with and without real-time requirements. Experimental results for parallel implementations of IEEE 802.11g and WCDMA signal processing algorithms provide a proof of concept.

Because the structure of targeted parallel programs as well as task deadlines is known a priori, it is possible to synthesize multiprocessor architectures that are optimum for the programs. It can be experimentally shown that such customized architectures are superior compared to domain-specific ones. That fact is important because embedded applications exhibit a wide diversity with respect to the complexity of their algorithms, the rate at which the algorithms need to operate, as well as their memory and intertask communication patterns. The consequence is that it is virtually impossible to find a good architecture that can meet the requirements of a wide range of algorithms. Customized architectures are therefore vital.

Since it is desirable to customize a system for a target embedded parallel application, automated tools for design space exploration are required to cope with the complexity. Whereas a skilled engineer can effectively utilize workbench-based tools [5, 6] to design a feasible architecture, the sheer number of design parameters renders a disciplined exploration infeasible. Further, it has been shown in [7] that often no consistent trend with respect to design objectives can be observed when design parameters are systematically changed, where the value of the objective function can increase or decrease by more than two orders of magnitude in an apparently random manner when moving between adjacent sets of parameters in the design space. Moreover, results obtained can be counter-intuitive. Consequently, even experienced designers cannot effectively execute a guided exploration based on their expertise because it is not easy to predict the outcome of parameter variation so that superior design points can be easily missed. Those results underline the need for an automated approach.

To enable an automatic exploration, design parameters and the objective must be mathematically modeled. Since this is inherently a combinatorial optimization problem,

it is natural to model the problem as such and solve it using Integer Linear Programming (ILP). However, the large number of design parameters that needs to be considered at the system level leads to a huge number of variables and constraints, thereby posing a serious challenge for ILP solvers that manifests itself in very long synthesis time. But to be useful, this automated synthesis approach must be fast in order to facilitate a systematic exploration.

On the other hand, recent advances in propositional satisfiability (SAT) methods [8] have spurred significant improvements in methods for Answer Set Programming (ASP) [9, 10]. ASP is a form of declarative programming oriented towards difficult search problems. Given the success that has been reported for solving such problems, it is interesting to study the effectiveness of these methods for speeding up the automated synthesis problem. Therefore, this paper compares ILP versus ASP-based high-level synthesis both in terms of synthesis runtime and the quality of synthesized architectures. The study uses parallel implementations of baseband signal processing chains for IEEE 802.11g and WCDMA wireless standards.

The rest of this paper is organized as follows. Summaries on related work and on our design flow are given in Sections 2 and 3, respectively. The ILP model for optimization is presented in Section 4, followed by a model of the problem in ASP semantics in Section 5. Finally, a comparison of the two methods and concluding remarks are given in Sections 6 and 7, respectively.

2. Related Work

Mathematical modeling and tool automation for synthesizing multiprocessor systems are an area that has been extensively discussed in the literature using techniques ranging from combinatorial optimization, through dynamic programming, simulated annealing, evolutionary algorithms to application-specific heuristics.

The vast majority of related work in this area have the drawback that the design space is pre constrained by fixing the architecture first, followed by task mapping and scheduling [11–13]. Since no customization is possible, resulting solutions are not optimum because the optimum application-specific system is imposed by the nature of the application, as dictated by operations performed within its tasks, as well as by intertask communication pattern. Often pre constraining is employed to overcome the complexity of the design space.

Some approaches attempt to reduce the design complexity by eliminating design dimensions [14, 15] thereby limiting the optimality of architectures. Advanced related works [16–18] recognize and address this aspect and attempt to solve the subproblems simultaneously. However, these approaches separately consider subproblems, effectively pre constraining the design space, albeit to a much smaller degree. Other approaches such as [19] randomly search for a feasible solution and may not lead to an optimum solution.

Scheduling during or after mapping too has been extensively treated in the literature [19, 20]. There also

have been efforts to map tasks in a way that optimizes for power [21], reduces chip temperature at run time [22], or minimizes interprocessor communications [23]. Dynamic mapping techniques have also been introduced with objectives such as temperature management [24] and performance optimization through adaptive mapping [25]. These approaches however consider scheduling on fixed architectures.

In contrast, in order to synthesize application-specific optimum architectures, it is important to simultaneously (i) select processors, (ii) map and schedule tasks to them, and (iii) select one or several networks for communications, such that design constraints and objectives are met. This avoids the problem of preconstriaining the design space, leading to globally optimum architectures.

The contribution of the method presented in this paper is a comprehensive mathematical model that can be used for automated design space exploration without limiting the design space as well as a comparison of candidate approaches to tackle the problem.

3. The Design Flow

Figure 1 depicts the flow. The input to the flow is a parallel program, and optionally information on task periods. The application is simulated and analyzed to obtain intertask data traffic and task precedence information. This information is used to specify an instance of an ILP problem or an ASP program. Similarly to other related work in this area, the other input to the design flow is information on available processing elements and communication networks, as well as their costs and constraints. In our approach, the design space is not pre-constrained, and the problem dimensions are not ranked. This ensures the optimality of found solutions. For real-time systems, it is often sufficient to meet timing constraints so that the interest is not to find the fastest solution. In such situations, the flow can be used to find the smallest system instead.

The solution generated by the ILP/ASP solver is used to generate an abstract description of the system, which is passed to further tool-chains described in [1] to generate the configuration bit stream. Because postsynthesis results could deviate from initial cost models used, new cost models can optionally be extracted after placement and routing to start a new iteration.

4. ILP Model

The ILP model used for automated synthesis in this work consists of two major parts. The first part covers constraints that establish the system functionality, without any regards to deadlines [7]. The second part covers scheduling and the optimization objective and is the focus of this paper.

The following notation is used. $I_i \in \{I_0, \dots, I_n\}$ is a task, $J_j \in \{J_0, \dots, J_m\}$ is a processor, and x_{ij} is a Binary Decision Variable (BDV). $x_{ij} = 1$ means that task I_i is mapped on processor J_j , $x_{ij} = 0$ otherwise.

The objective function for a terminating parallel program, or for one period of a non-terminating program, is expressed as

$$\min \left(\sum_{i=0}^n \sum_{j=0}^m x_{ij} \cdot T_{ij} + T_{\text{net}} + T_{\text{switch}} \right), \quad (1)$$

where T_{ij} is the execution time of task I_i on processor J_j , and T_{net} is the cost in time of using communication resources as described in [7]. T_{switch} is the cost in time of task switching which depends on several factors as described in subsequent sections

$$T_{\text{switch}} = \sum_{l=0}^{2^n-2} \sum_{j=0}^m \gamma_{lj} \cdot F_{lj} \cdot (T_{s_{lj}} \cdot t_j + O_j). \quad (2)$$

4.1. The Processor Architecture. This is the actual cost of switching context, which depends on the memory and on the microarchitecture, as well as on the mechanism for context switching (i.e., under software or with hardware support). The coefficient t_j captures this cost and can be reliably precomputed for processors of interest. Whether this cost is actually incurred depends on task mapping as discussed in Section 4.4.

4.2. The Kernel/OS. The kernel or real-time OS introduces a control overhead due to scheduling (polling, moving tasks between run and delay queues, etc.). In this formulation, it is assumed that kernel/OS is already selected and is fixed, for each of the processors in the design space (i.e., OS selection is not a part of the optimization problem, so that the associated cost is coupled to the selected processor). This is however not a limitation because, when desired, instances of the same processor running different operating systems or microkernels can be specified in the ILP problem to extend the design space.

The overhead is caused by the clock interrupt handler interfering the execution of application tasks because of its higher priority. This increases the number of task switching and the response time of application tasks. The coefficient O_j in (2) captures the latter cost, the clock-handler time. The analysis in [26] describes how the clock-handler time can be estimated for fixed-priority schedulers. Because the overhead is kernel/OS-specific, and may depend on task mapping, the computation/estimation of O_j in the problem formulator (Figure 1) is implemented in an extensible way to support new kernel/OS models.

4.3. The Schedule/Task Switching. The schedule determines how often task switching takes place as captured by the coefficient $T_{s_{lj}}$ in (2). $T_{s_{lj}}$ is the number of task switching that is incurred for the duration of the application, or for one period, when a particular group of tasks with the index l is mapped on a processor J . One can distinguish between three major scheduler categories: cooperative, fixed-priority preemptive, and deadline-driven schedulers.

In simple cooperative schedulers (e.g., cyclic executives), there is no preemption, so that $T_{s_{lj}} = 0$. The overhead

FIGURE 1: Architecture synthesis flow.

incurred when a task begins and ends is already captured by T_{ij} in (1) as part of function-call overhead.

Preemptive schedulers often require task priorities to make decisions by letting higher-priority tasks run first. This can improve performance if critical tasks are assigned higher priorities. Priorities can be fixed or dynamic depending on whether priorities can change at runtime. The exception to this distinction are measures against priority inversion. Even though such measures do change priorities dynamically, the changes are temporary to otherwise fixed priorities.

Deadline-driven schedulers change task priorities dynamically and have the advantage that deadlines can be guaranteed at higher CPU utilization compared to fixed-priority scheduling. In either case, T_{sij} is a function of the number of context switching, and its usage in the ILP model is the same. This section discusses how the worst-case number of context switching can be estimated for fixed-priority schedulers, which are more typical in real-time embedded systems.

For these schedulers, T_{sij} is equal to the number of interferences due to higher priority tasks and is obtained from Rate Monotonic Analysis (RMA) [27] within the problem formulator. The RMA in the formulator currently

supports tasks with single deadlines, and which have fixed durations and nonvarying periods. However, the implementation is easily extensible to support flexible RMA models. Such models can be adapted to applications with arbitrary, multiple, or internal deadlines [28]. Future extensions will affect the computation of T_{sij} only. The ILP model for synthesis remains unaffected.

RMA is conducted for all possible task groups and mappings. The output of RMA, the response r , is used to estimate T_{sij} . Algorithm 1 shows how the parameter is computed.

The first line computes a scheduling table for a group G_l of tasks, if the group would be mapped on a processor J_j . The rows in the scheduling table contain the priority of a tasks in the group, together with their deadlines, periods and execution times on the processor. The priorities are computed according to [27].

The table is initially filled in arbitrary order with task information, and the priorities are initially zero. The rows are then sorted in two passes according to periods and deadlines.

Prior to sorting, deadlines are relaxed according to Algorithm 2 in order to avoid pessimistic schedulability analysis. The analysis assumes that all tasks are released at the same time. If the response of a task is then greater than

```

(1) Table  $T = \text{createSchedulingTable}(G_l, J_j,$ 
    task time  $T_{ij}$ , task deadlines  $D_i$ , task periods  $\mathcal{T}_i)$ 
(2) for all  $I_i \in G_l$  do
(3)   Response  $r_i = \text{computeResponse}(I_i, T)$ 
(4)    $F_{ij} = 1$ 
(5)   if  $r > D_i$  then
(6)      $F_{ij} = 0$ 
(7)   end if
(8) end for
(9)  $r = \text{computeLargestResponse}(G_l, T)$ 
(10)  $T_{sij} = \lceil r / \text{period of highest priority task in } G_l \rceil$ 

```

ALGORITHM 1: Determining context switching cost.

```

(1) for each  $I_i \in G_l$ , set parent deadline to zero
(2) while  $G_l$  is not empty do
(3)   if  $G_l$  is circular then
(4)     select  $I_i \in G_l$  such that  $\mathcal{T}_i \leq \mathcal{T}_{i_2}$  for all  $I_{i_2} \in G_l$ 
(5)   else
(6)     select  $I_i \in G_l$  with no parent in  $G_l$ 
(7)   end if
(8)   for each child  $I_{i_2}$  of  $I_i$  in  $G_l$  do
(9)     if parent deadline  $< D_i$  then
(10)      set parent deadline to  $D_i$ 
(11)    end if
(12)   end for
(13)   if  $I_i$  had a parent in  $G_l$  then
(14)      $D_i +=$  parent deadline
(15)   end if
(16)   remove  $I_i$  from  $G_l$ 
(17) end while

```

ALGORITHM 2: Deadline relaxation during RMA.

the deadline as shown in Algorithm 1, the schedule is declared infeasible. However, if there is a precedence relationship, not all tasks are released at the same time. In particular, if there is an edge between I_{i_1} and I_{i_2} , then I_{i_2} cannot start until I_{i_1} has finished. Therefore, the deadline D_{i_2} needs to be relaxed to $D_{i_2} = D_{i_2} + D_{i_1}$ to reflect the fact that there is an offset from the release time of its parent task.

Relaxation proceeds by selecting the most critical task. If the subgraph G_l of the application graph G is circular, it is not immediately obvious which task is most critical because of circular producer-consumer relationships. Therefore, the algorithm selects the task in G_l with the shortest deadline. Because a critical task is eliminated from G_l at the end of each iteration, this selection has the effect of introducing cuts in G_l which removes circular paths. Otherwise, if G_l has no cycles, the most critical task in G_l is the one that does not consume data from other tasks in the subset. Before a task is removed from G_l , its deadline is relaxed by adding the deadline of its already removed parent, if the task *had* one. If the task had multiple parents, then the largest of its parents' deadlines is selected according to lines 8–12. This relaxation does not

impose any limitation to the type of application graph that can be handled by the synthesis flow.

After relaxation, sorting begins. The first pass sorts tasks according to periods in ascending order. If the tasks have different periods, and G_l is at least partially connected, then a critical assumption is made that *if there is a node in G_l with a period less than that of any of its parent, then the edge between the node and the parent represents a weak precedence meaning that the corresponding task can execute without receiving data from its parent*. An example would be a task that infrequently obtains new parameters from another task for its internal computations. Otherwise, the application graph is faulty, and the resulting schedule is meaningless. Partial connectedness in this context means that G_l contains at least one nontrivial connected subgraph.

The second pass sorts the table again according to deadlines, but the sorting is done only within rows containing the same period. Since deadlines have been previously relaxed, no distinction with respect to precedence relationship between tasks needs to be made; if tasks I_{i_1} and I_{i_2} have no direct or indirect precedence, then I_{i_1} must finish before I_{i_2} if $D_{i_1} < D_{i_2}$, because I_{i_1} needs to finish earlier; if there is a precedence relationship, then $D_{i_1} < D_{i_2}$ because of the relaxation step, and I_{i_1} must appear before I_{i_2} . Indirect precedence in this context means that there is a path from I_{i_1} to I_{i_2} via one or more intermediate tasks. Therefore, because second sorting is only done within rows, tasks with shorter periods appear before those with longer periods regardless of whether or not latter tasks have shorter deadlines.

The sorting is topological and is thus not unique. Moreover, if the group represents a nonconnected graph, then the result after sorting is a partial order. The final order after sorting reflects the priorities in descending order, which are assigned by a simple enumeration.

With the table in place, the algorithm proceeds to compute the response time of each task in the group G_l according to the scheduling table. The response is computed recursively according to [27] as

$$r_i = T_{ij} + \sum_{\forall I_{i_h} \in G_l | \text{priority}(I_{i_h}) > \text{priority}(I_i)} \left\lceil \frac{r_i}{\mathcal{T}_{i_h}} \right\rceil \cdot T_{i_h j}. \quad (3)$$

This model of response time differs slightly from that of Liu and Layland [27] in that no bound in task blocking time due to safeguarding against priority inversion is included. This is because the programming model used here is message passing so that tasks do not share protected data such that semaphore-based synchronization for variables or memory locations is not required.

The analysis then concludes by comparing the response time against the execution time in lines 4–7 of Algorithm 1. The scheduling feasibility parameter F_{ij} in (2) is set to 0 if the response time is larger. Finally, the number of task switching is estimated in line (10) from the response time of the lowest-priority task and the period of the highest-priority task. This worst-case estimate is conservative by making the assumption that all tasks in the group are always ready when released so that the lowest-priority task experiences

maximum interference. The use of the parameter F_{lj} is explained in the following subsection.

4.4. Task Mapping. Task mapping influences the switching costs in two ways: (i) by selecting the processor, the switching mechanism, and thus the cost, is determined and (ii) by grouping tasks on one processor, the optimum schedule that can be applied, and thus the number of task switching, is determined. Consequently, scheduling and task-mapping influence each other during optimization.

To include this cross-effect during optimization, two strategies can be followed: (i) integrate scheduling in an ILP solver so that a schedule is computed prior to cost calculation for a candidate mapping or (ii) precompute optimum schedules for all possible mappings, and integrate the schedules in the ILP formulation. In this work, we opted for the latter strategy as discussed in the previous subsection. This is because, by pre-computing the schedules, infeasible mappings can be eliminated to reduce the size of the ILP instance. For this purpose, a coefficient F_{lj} is used in the formulation in (2). This coefficient is computed in the ILP formulator during RMA. Its value is 1 if there is a feasible schedule for a group of tasks with the index l on processor J , and 0 otherwise. We next describe how F_{lj} is used to enforce feasibility constraints in the formulation.

Let $\mathcal{P}(I)$ be the power set of the task set $I = \{I_0, \dots, I_n\}$. Let G_l be an element in the power set excluding the empty set, with $l = \{0, 1, \dots, 2^n - 2\}$. Let $I_{\text{mapped}} \subset \mathcal{P}(I)$ be a set, so that each element contains one or more tasks that will be mapped on the same processor. The solution to the combinatorial optimization problem consists of the set I_{mapped} . Each element in I_{mapped} is associated with a task switching overhead as dictated by its schedule.

Now since I_{mapped} is not known at formulation time, a variable $\mathcal{M}_{lj}$ is introduced for each element G_l in the power set $\mathcal{P}(I)$. If $\mathcal{M}_{lj} = 1$, then $G_l \in I_{\text{mapped}}$. If $\mathcal{M}_{lj} = 0$, then G_l is not an element of I_{mapped} . Therefore, we insist that

$$\mathcal{M}_{lj} \leq F_{lj} \quad \forall \mathcal{M}_{lj}, \quad (4)$$

so that if and only if the mapping is feasible, then $\mathcal{M}_{lj}$ constitutes a degree of freedom during synthesis. We next describe how the decision variables $\mathcal{M}_{lj}$ and x_{ij} are linked through ILP constraints.

Recalling that $x_{ij} = 1$ implies that a task I_i is mapped on a processor J_j , it follows for any group G_l , $\mathcal{M}_{lj} = 1$ if and only if $x_{ij} = 1$ for all $I_i \in G_l$. This results into a logical constraint

$$\mathcal{M}_{lj} = (x_{i_{0j}} \wedge x_{i_{1j}} \wedge \dots \wedge x_{i_{lgj}}) \quad \forall \mathcal{M}_{lj} \quad (5)$$

$$\text{with } G_l = \{I_{i_0}, I_{i_1}, \dots, I_{i_{lg}}\}, \quad lg = |G_l| - 1.$$

To transform the logical constraint into a linear form, two steps were applied. First, we specified that

$$\begin{aligned} \mathcal{M}_{lj} = 0 &\longrightarrow (x_{i_{0j}} + x_{i_{1j}} + \dots + x_{i_{lgj}}) < |G_l|, \\ \mathcal{M}_{lj} = 1 &\longrightarrow (x_{i_{0j}} + x_{i_{1j}} + \dots + x_{i_{lgj}}) = |G_l|. \end{aligned} \quad (6)$$

These two constraints insure that when a schedule is not feasible, then at least one $I_i \in G_l$ is not mapped on J_j . This implies that other groups which are either proper subsets of G_l , or which are not super sets of G_l , can be mapped on J_j , provided that they have a feasible schedule. The second step then is a set of inequalities that satisfy the specification in (6)

$$x_{i_{0j}} + x_{i_{1j}} + \dots + x_{i_{lgj}} \leq |G_l| - 1 + \mathcal{M}_{lj}, \quad (7)$$

$$x_{i_{0j}} + x_{i_{1j}} + \dots + x_{i_{lgj}} \geq |G_l| \cdot \mathcal{M}_{lj}. \quad (8)$$

With (4), (7), and (8), feasible mappings are guaranteed. The last step is to capture the switching cost of the groups in the objective. A contribution of a group to the switching cost is given by $\mathcal{M}_{lj} \cdot T_{s_{lj}}$. However, this contribution cannot be directly used in the objective function by taking the sum of all contributions from all groups. This is because, as it can be observed from (7) and (8), if a group G_l is mapped on a processor J_j , then the value of $\mathcal{M}_{lj}$ for all groups which are subsets of G_l is also one. Consequently, taking the sum of contributions directly would erroneously include the switching cost of subgroups.

To prevent this incorrect inclusion of switching costs, we need to specify that the switching contribution of a group should be counted only when $\mathcal{M}_{lj} = 0$ for all of its supersets. This leads to nonlinear terms in the objective function of the form $\mathcal{M}_{lj}(\mathcal{M}_{l_{s1j}} - 1)(\mathcal{M}_{l_{s2j}} - 1) \dots$, where l_{s1} and l_{s2} are indices of supersets for a group with an index l .

To break the non-linearity, a BDV γ_{lj} is introduced for each $\mathcal{M}_{lj}$. Its value is 1 only when the group is mapped, and $\mathcal{M}_{lj} = 0$ for all supersets. To model this property, we note from (8) that the left-hand side of the inequality for a group G_{l_1} with $\mathcal{M}_{l_1j} = 1$ is greater than the left-hand side of the same inequality for a group G_{l_2} , if $G_{l_2} \subset G_{l_1}$. Therefore, the following relationship holds:

$$0 \leq \frac{1}{|I|} \left(\sum_{I_i \in I} x_{ij} - \sum_{I_i \in G_l} x_{ij} \right) < 1 \quad \forall G_l, J_j. \quad (9)$$

The first sum in the above relationship is the total number of tasks that have been mapped on J_j . The second sum is the size of a group, which is the same as the left-hand side of (8). The difference of the two sums is zero in two cases: (i) if nothing is mapped on J_j and (ii) if a mapped group has no superset G_{l_s} for which $\mathcal{M}_{l_sj} = 1$ for that specific mapping. The sum is greater than zero, if for G_l , there is a superset G_{l_s} with $\mathcal{M}_{l_sj} = 1$. This is because there is at least one decision variable x_{ij} with value 1 in the first sum, which is not present in the second. The largest value that the difference of the two sums can have is $|I| - 1$, so that the upper limit in (9) is 1.

We next exploit this relationship by specifying that

$$\gamma_{lj} + \frac{1}{|I|} \left(\sum_{I_i \in I} x_{ij} - \sum_{I_i \in G_l} x_{ij} \right) \leq 1, \quad (10)$$

$$\gamma_{lj} + \frac{1}{|I|} \left(\sum_{I_i \in I} x_{ij} - \sum_{I_i \in G_l} x_{ij} \right) \geq \mathcal{M}_{lj} - 1 + \frac{1}{|I|}.$$

4.5. Processor-External Factors. Processor-external factors such as interrupts and data availability have a direct runtime impact on the schedule. The foregoing formulation has the limitation that it is based on worst-case assumptions in rate monotonic analysis. In particular, it is assumed that tasks in a group G_l with no precedence relationship can become ready at the same time.

With respect to data availability, the worst-case assumptions can be relaxed by taking into account in RMA when data can actually arrive depending on source-task-destination-task mapping, and on the selected communication network.

A possible relaxing solution is to compute offsets between release times of tasks with indirect precedence. For example, if there are three tasks such that the first sends data to the second, and the second to the third, and the mapping is such that the first and third are mapped on one processor, and the second on another, then there is an offset between release times of the first and third tasks. This offset is equal to the time needed for data to be sent to the second task, plus the response time of the second task, plus the time for the resulting data to be sent from the second task to the third task. A suitable ILP formulation that will not significantly increase the problem size needs to be found.

5. Answer Set Programming

Answer Set Programming (ASP) is different from procedural programming in that a problem is described using a formal language, and a solver finds a solution. A problem is presented as a logic program consisting of a set of atoms and rules [29]. An atom is a Boolean proposition about the problem universe; whereas rules specify relationships between the atoms. A solution to a program is called a stable model and tells which atoms are true [29]. This is similar to SAT problems if rules and stable models are perceived to be clauses and satisfying assignments, respectively. ILP models can be encoded into equivalent ASP programs. We opted to use the ASP solver clasp [9] whose grounding tool natively supports the encoding of linear constraints [30]. The native support eliminates the need of having to translate constraints into clauses, a procedure that can lead to a huge number of clauses [31]. The rest of this section describes how the ILP model from Section 4 is coded into an equivalent ASP program. The same notation is used so that variables from Section 4 now stand for atoms.

Linear inequalities are coded into rules whose general form is

$$b[v_0 = a_0, v_1 = a_1, \dots, v_n = a_n]c, \quad (11)$$

where the syntax $v_i = a_i$, $a_i \in \mathbb{N}$ denotes the weight a_i of a variable v_i in a linear (in)equality, and the syntax $b[\dots]c$, $b, c \in \mathbb{N}$ is a general form for constraining such that a coded constraint represents an equality if $b = c$, a less-than inequality if b is not specified (is absent), and a greater-than inequality if c is not specified [30]. Since weights in the ILP model are generally out of $\mathbb{R}$ whereas $a_i, b, c \in \mathbb{N}$, rounding is required. We therefore round a_i

and b up, and c is rounded down. Consequently, more restrictive constraints result, which can theoretically exclude a solution that otherwise does not violate the original problem constraints. This is the reason that we compare the quality of generated solutions in Section 6.

The general form (11) is used throughout for constraints, with a few exceptions in which constraints can be directly expressed as clauses, and thus directly as ASP rules. This has the advantage of eliminating auxiliary variables and associated constraints such that the overall problem size becomes smaller. An example is (5); the link between the auxiliary variable $\mathcal{M}_{lj}$ for a group of tasks and mapping decision variables x_{ij} is already in conjunctive form so that the constraint can be specified directly as a rule

$$\mathcal{M}_{lj} \leftarrow x_{i_0j}, x_{i_1j}, \dots, x_{i_{lg}j}, \quad (12)$$

thereby dropping (7) and (8). However, (12) represents a logical implication, whereas (5) is a logical equality. Without further measures, a stable model can potentially have $\mathcal{M}_{lj}$ as true when one or several of the atoms $x_{i_{lg}j}$ are false. We therefore additionally add the rule

$$\leftarrow \mathcal{M}_{lj}, 1[\text{not } x_{i_0j}, \text{not } x_{i_1j}, \dots, \text{not } x_{i_{lg}j}], \quad (13)$$

as an integrity constraint [30] such that $\mathcal{M}_{lj}$ is not derived if any of its associated atoms is not derived.

Similarly, while not directly obvious, (10) stand for logical conjunctions that can be represented by the rule

$$\gamma_{lj} \leftarrow \mathcal{M}_{lj}, \text{not } \mathcal{M}_{l_{s_0}j}, \text{not } \mathcal{M}_{l_{s_1}j}, \dots, \quad (14)$$

where $\mathcal{M}_{l_{s_i}j}$ is the i th superset of the group G_l . The implication is that γ_{lj} is derived when the corresponding group $\mathcal{M}_{lj}$ is derived, but none of the atoms for associated supersets is derived.

Using the same syntax for specifying weights, the objective function has the form [30]

$$\text{minimize } [v_0 = a_0, v_1 = a_1, \dots, v_n = a_n], \quad (15)$$

but in this case the weights a_i are not directly rounded, rather, the weights which represent costs in time are converted into processor cycles so that (15) matches (1) as close as possible. In order to avoid large numbers which can overflow the computation of the objective function, these weights are expressed in terms of cycles that would have been spent on the slowest processor, normalized by the number of cycles on the same processor that would have been consumed for the smallest weight in the objective.

6. Comparison of Synthesis Results

This section compares synthesis runtime as well as quality of results for ASP-based synthesis against the ILP-based flow. For this purpose, two parallel programs have been implemented using the Message Passing Interface (MPI) standard [32]. Only a subset of the standard that can be efficiently implemented in embedded systems has been used.

TABLE 1: WLAN tasks.

Function	Index	Deadline	P1	P2	P3
Master	I0	19	6032	1034	11
Phase	I1	988	3504	601	601
Channel	I2	988	909	156	156
Timing	I3	19	33781	5791	17
Demap	I4	1216	598	102	102
FFT	I5	1216	534	534	534
Fine	I6	19	11	11	11
Coarse	I7	19	13	13	13

TABLE 2: WCDMA tasks.

Function	Index	Deadline	P4	P5
Master	I0	65	4376	43
FFT	I1	4160	3788	3788
Vector mul	I2	4160	3082	3082
FFT	I3	4160	3788	3788
ov. add	I4	4160	3665	3665
Long Code	I5	1040	3784	147
DLL	I6	1040	8746	1000
Short Code	I7	1040	835	143
Rake	I8	1040	16204	686
Combiner	I9	16640	383	383

The first application implements a signal processing chain for IEEE 802.11g WLAN standard between the antenna and the channel decoder. The algorithms are described in [33–36]. The second application implements the functionality for the same portion for WCDMA. The algorithms are described in [37–39]. Tables 1 and 2 show the parallel tasks and their deadlines in nanoseconds. In this implementation, the deadlines are equal to the periods. The latter were obtained from the standards. The three last columns in Table 1 show approximated execution time in nanoseconds of the tasks on three different processors with loosely coupled accelerators. The execution times were obtained by executing the tasks on given processors. Similarly, the two last columns in Table 2 show the approximated time for WCDMA tasks on two other processors.

The differences in execution times in the two tables are caused by different type of accelerators attached to the processors. For example, all cores have a butterfly accelerator based on the architecture described in [40]; therefore, the execution time for FFT tasks is the same for all processors (the difference for FFT tasks between WLAN and WCDMA is due to the different number of FFT points). On the other hand, only P2 and P3 have a CORDIC accelerator; therefore, the execution time for the carrier-phase offset estimation and compensation task is only 156 nanoseconds for these cores, versus 909 nanoseconds for P1.

Tables 3 and 4 show the communication traffic between tasks for the two applications. These were obtained from MPI simulations for 27 OFDM symbols for WLAN, and for one slot for WCDMA. The third and fourth columns show the amount and number of data transfers, respectively.

TABLE 3: Traffic pattern for WLAN.

Src (i_1)	Dst (i_2)	$D_{i_1 i_2}$ (bytes)	$B_{i_1 i_2}$
I0	I2	220	55
I0	I7	2339608	2242
I0	I1	220	55
I4	I2	5824	28
I4	I1	5824	28
I6	I5	7168	28
I2	I1	5824	28
I7	I6	2913300	2241
I5	I4	7168	28
I1	I3	5824	56

In order to additionally compare the synthesis runtime with and without fixed-priority preemptive scheduling, configurations with a basic cyclic executive and with a preemptive kernel were used for all processors with each run.

For ILP, the same solver settings were used for all cases (node auto-ordering, most feasible basis crash, automatic branch and bound branching, and presolving of rows and columns) [41].

No options were specified for ASP. However, we determined that splitting the objective into its three constituent parts for execution, communication, and scheduling time significantly speeds up the ASP solver time by up to two orders of magnitude. This circumstance was exploited subsequently since the speed up is not accompanied by any penalty in quality of the solution. Splitting the objective function is a feature in clasp/clingo that was conceived to avoid possible overflows when computing the value of the objective function because of integrality of weights [30].

Tables 5 and 6 show the parameters of used processors and networks, respectively. The number of processors and networks used was 6 in each case. Whereas the number of resources could be selected to reflect the number of nodes and edges in the application graph, it is advantageous to start with a smaller number to speedup the synthesis. If resources are exhausted, the number should be increased in another run to avoid preconstriaining the design space.

Table 7 summarizes the results, which were obtained on a machine with a T5500 processor and 2 GB of memory. The columns “No. cons.” show the number of constraints and the number of rules for ILP and ASP modes, respectively. Similarly, the columns “No. var.” show the number of decision variables and atoms for the two modes. These numbers give a measure for the complexity of the problem instances. The number of variables for ILP mode is much less than the number of atoms for ASP mode because the ILP solver, *lp_solve*, has a presolve option that can reduce the size of the problem by eliminating redundant constraints. This option was exploited because presolving tends to reduce the solver time.

The columns “Form.” and “Solver” show synthesis runtime spent formulating and solving the problem, respectively. Formulator time is rather large; most of this time is spent reading text files generated from MPI simulations.

TABLE 4: Traffic pattern for WCDMA.

Src (i_1)	Dst (i_2)	$D_{i_1 i_2}$ (bytes)	$B_{i_1 i_2}$
I8	I9	40948	2560
I1	I2	81924	161
I0	I1	81412	160
I5	I8	10236	2559
I5	I6	40960	10240
I4	I6	40964	10241
I3	I4	81924	161
I2	I3	81924	161
I6	I8	81892	5119
I7	I8	10236	2559

TABLE 5: Parameters of used processors.

Name	T_j (μs)	A_j (slices)	No. instances
P1	0.70	450	2
P2	0.93	526	2
P3	1.20	605	2
P4	1.20	553	3
P5	1.19	724	3

TABLE 6: Parameters of used networks.

Type	M_k	L_k (μs)	p_k	τ_k (μs)	No. instances
Link	2	6.25	0	0	3
Bus	16	8.00	0.0625	16	3

The sizes of the files in these experiments were 4.8 GB and 6.4 GB for WCDMA and WLAN, respectively. They contain, among others, time stamps for each data packet transmitted between tasks. This large time is not a limitation for automated exploration; much faster time can be achieved by using compressed binary files and/or usage of cache files to capture relevant information only during automated explorations.

The solver times for ASP mode are dramatically shorter by up to three orders of magnitude. Given that ASP solver time is in the order of few seconds (versus up to 8 hours for ILP mode), synthesizing using ASP is a promising approach for automatically exploring a large number of design alternatives as it is the requirement for this flexible multiprocessor synthesis problem.

As previously mentioned, we additionally need to compare the quality of results because of a potential for over-constraining in ASP mode, particularly in light of the fact that ASP mode is much faster. The columns “Obj.” show the value of the objective function after optimization. Since both methods are heuristic, it is interesting to know how far integer solutions are away from corresponding relaxed solutions, which is a measure of how good a solution is in case the solver times out. A timeout occurred once for WCDMA for ILP synthesis mode under cyclic scheduling. The columns “Gap” give this measure, which indicates that the solution was quite good even in the timeout case.

FIGURE 2: Synthesized architectures under preemptive scheduling. Dashed lines: links, full lines: buses.

Comparing the value of the objectives for the two modes, the differences before rounding are insignificant with the exception of the timeout case where ASP mode found a better solution. The impact of more restrictive constraints for ASP mode due to rounding as discussed in Section 5 was not apparent in these experiments. While these particular results are suggestive, experiments with a much larger set of parallel programs are still required to characterize the potential impact.

The impact of scheduling constraints can be seen by comparing the values of the objective functions under the two scheduling modes. Better values were obtained under cyclic scheduling because no deadlines were imposed when mapping tasks so that only the execution times needed to be considered. Consequently, the solvers attempt to group tasks such that expensive intertask communications are minimized. However, these apparently faster architectures are practically not usable because of no deadline guarantees.

Figure 2 shows synthesized architectures under the two modes for preemptive scheduling. These architectures are very similar for WLAN, and the same architecture was obtained for WCDMA. This result emphasizes the potential for ASP-based synthesis, since the quality of results was not traded against solver runtime. Synthesized architectures are not necessarily unique because there may exist several optimum solutions to a combinatorial problem. Thus, for WLAN case, where the two architectures are similar but not the same, it is quite possible that either architecture could have been obtained through either of ILP or ASP synthesis mode. This is because all resources are allocated

TABLE 7: Synthesis results. Nonshaded rows for ILP, shaded rows for ASP.

Appl.	Cyclic						Preemptive					
	Problem size		Run time (sec)		Obj.	gap	Problem size		Run time (sec)		Obj.	Gap
	No. cons.	No. var.	form.	solver	(sec)		No. cons.	No. var.	form.	solver	(sec)	
WLAN	7089	3638	497	636	0.0001	0.0	5901	2538	552	1010	0.10	9.6
WCDMA	32261	13320	701	28811	18.25	7.8	25394	12792	704	19567	16.90	0.2
WLAN	7033	207412	1467	2.421	0.0002	—	7033	207412	1477	16.797	0.08	—
WCDMA	25631	21806	977	6.781	16.898	—	25631	21806	1125	10.219	16.90	—

through a non-constraining combinatorial optimization process according to (1) based on resource parameters and characteristics of the parallel program, and not on the synthesis method used.

Finally, two discussion points are in order for these architectures. First, as previously mentioned, our proposed design flow does not pre-constrain the design space. As a result, multiple communication resources are allocated between any pair of processors in this experiment. This allocation minimizes expensive intertask communications which are necessary under preemptive scheduling because of schedulability constraints leading to conditions such that certain tasks cannot be mapped on the same processor. Thus, the generated architecture description (Figure 1) includes not only the netlist, but also information on which communication libraries a task should use to communicate with any other task. This information is used by configuration tools to automatically bind appropriate low-level communication libraries for different networks, links and buses.

Second, the physical implementations of messaging passing interfaces make use of FIFO to queue messages. This means that, once a task has initiated a data transfer by calling the appropriate function, the task is free to do further processing and to initiate or wait for data via another communication resource. The result is that communication latencies can be hidden through overlapping. However, this effect is not accounted for in the objective function because temporal information is not used. Consequently, the actual cost in total computation time can be smaller than what the objective function indicates.

Accounting for temporal information is not a feasible prospect because a far greater number of variables would need to be considered to model and capture all possible moments in which communications can be initiated.

7. Summary and Conclusion

In this paper, a method for automated architecture synthesis for FPGA multiprocessor systems has been presented. The method takes into account fixed-priority preemptive scheduling to cover a broad spectrum of embedded application requirements. Combinatorial optimization is used during synthesis. A case study, in which architectures for IEEE 802.11g and WCDMA baseband signal processing algorithms are synthesized, demonstrates the feasibility of the automated synthesis by showing that problems with sizes that can be encountered in the embedded domain can be

solved. Synthesis based on ILP and ASP methods has been compared. ASP mode has a far greater potential for solving difficult synthesis problems. Solver times in this mode were in the order of a few seconds, which is up to three orders of magnitude faster compared to ILP-based synthesis without sacrificing the quality of results.

References

- [1] C. Bobda, T. Haller, F. Muehlbauer, D. Rech, and S. Jung, "Design of adaptive multiprocessor on chip systems," in *Proceedings of the 20th Symposium on Integrated Circuits and System Design (SBCCI '07)*, pp. 177–183, Rio de Janeiro, Brazil, September 2007.
- [2] W.-T. Zhang, L.-F. Geng, D.-L. Zhang, et al., "Design of heterogeneous MPSoC on FPGA," in *Proceedings of the 7th International Conference on ASIC (ASICON '07)*, pp. 102–105, Guilin, China, October 2007.
- [3] M. Saldana, D. Nunes, E. Ramalho, and P. Chow, "Configuration and programming of heterogeneous multiprocessors on a multi-FPGA system using TMD-MPI," in *Proceedings of the IEEE International Conference on Reconfigurable Computing and FPGA's (ReConFig '06)*, pp. 1–10, San Luis Potosi, Mexico, September 2006.
- [4] N. Njoroge, J. Casper, S. Wee, et al., "ATLAS: a chip-multi-processor with transactional memory support," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '07)*, pp. 3–8, San Jose, Calif, USA, 2007.
- [5] T. Kangas, P. Kukkala, H. Orsila, et al., "UML-based multiprocessor SoC design framework," *ACM Transactions on Embedded Computing Systems*, vol. 5, no. 2, pp. 281–320, 2006.
- [6] S. Pasricha, N. Dutt, and M. Ben-Romdhane, "Extending the transaction level modeling approach for fast communication architecture exploration," in *Proceedings of Design Automation Conference (DAC '04)*, pp. 113–118, 2004.
- [7] H. Ishebabi and C. Bobda, "Automated architecture synthesis for parallel programs on FPGA multiprocessor systems," *Microprocessors and Microsystems*, vol. 33, no. 1, pp. 63–71, 2009.
- [8] M. R. Prasad, A. Biere, and A. Gupta, "A survey of recent advances in SAT-based formal verification," *International Journal on Software Tools for Technology Transfer*, vol. 7, no. 2, pp. 156–173, 2005.
- [9] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub, "clasp: a conflict-driven answer set solver," in *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR '07)*, vol. 4483 of *Lecture Notes in Computer Science*, pp. 260–265, Tempe, Ariz, USA, May 2007.
- [10] E. Giunchiglia, Y. Lierler, and M. Maratea, "Answer set programming based on propositional satisfiability," *Journal of Automated Reasoning*, vol. 36, no. 4, pp. 345–377, 2006.

- [11] D. Bertozzi, A. Jalabert, S. Murali, et al., "NoC synthesis flow for customized domain specific multiprocessor systems-on-chip," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 2, pp. 113–129, 2005.
- [12] A. Hansson, K. Goossens, and A. Rădulescu, "A unified approach to constrained mapping and routing on network-on-chip architectures," in *Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '05)*, pp. 75–80, Jersey City, NJ, USA, September 2005.
- [13] B. H. Meyer and D. E. Thomas, "Rethinking automated synthesis of MPSoC architectures," in *Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS '07)*, pp. 1–6, March 2007.
- [14] B. K. Dwivedi, A. Kumar, and M. Balakrishnan, "Automatic synthesis of system on chip multiprocessor architectures for process networks," in *Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and Systems Synthesis (CODES+ISSS '04)*, pp. 60–65, Stockholm, Sweden, 2004.
- [15] Y. Jin, N. Satish, K. Ravindran, and K. Keutzer, "An automated exploration framework for FPGA-based soft multiprocessor systems," in *Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '05)*, pp. 273–278, Jersey City, NJ, USA, September 2005.
- [16] C. Zhu, Z. Gu, R. P. Dick, and L. Shang, "Reliable multiprocessor system-on-chip synthesis," in *Proceedings of the 5th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '07)*, pp. 239–244, Salzburg, Austria, 2007.
- [17] C. Lee and S. Ha, "Hardware-software cosynthesis of multitask MPSoCs with real-time constraints," in *Proceedings of the 6th International Conference on ASIC (ASICON '05)*, vol. 2, pp. 919–924, Shanghai, China, 2005.
- [18] S. Ha, C. Lee, Y. Yi, S. Kwon, and Y.-P. Joo, "Hardware-software codesign of multimedia embedded systems: the PeaCE approach," in *Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA '06)*, pp. 207–214, Sydney, Australia, August 2006.
- [19] D. L. Rhodes and W. Wolf, "Co-synthesis of heterogeneous multiprocessor systems using arbitrated communication," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '99)*, pp. 339–342, San Jose, Calif, USA, November 1999.
- [20] P. Pop, *Analysis and synthesis of communication-intensive heterogeneous real-time systems*, Ph.D. thesis, Linköping University, Linköping, Sweden, 2003.
- [21] C. Erbas, S. Cerav-Erbas, and A. D. Pimentel, "Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 3, pp. 358–374, 2006.
- [22] M. Bao, A. Andrei, P. Eles, and Z. Peng, "Temperature-aware task mapping for energy optimization with dynamic voltage scaling," in *Proceedings of the 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS '08)*, pp. 1–6, April 2008.
- [23] M. Ruggiero, A. Guerri, D. Bertozzi, F. Poletti, and M. Milano, "Communication-aware allocation and scheduling framework for stream-oriented multi-processor systems-on-chip," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '06)*, vol. 1, pp. 3–8, Munich, Germany, 2006.
- [24] S. Carta, F. Mereu, A. Acquaviva, and G. De Micheli, "MiGra: a task migration algorithm for reducing temperature gradient in multiprocessor systems on chip," in *Proceedings of International Symposium on System-on-Chip Proceedings (SOC '07)*, Tampere, Finland, November 2007.
- [25] P. K. Holzenspies, J. L. Hurink, J. Kuper, and G. J. Smit, "Run-time spatial mapping of streaming applications to a heterogeneous multiprocessor system-on-chip (MPSoC)," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '08)*, pp. 212–217, Munich, Germany, March 2008.
- [26] A. Burns, K. Tindell, and A. Wellings, "Effective analysis for engineering real-time fixed priority schedulers," *IEEE Transactions on Software Engineering*, vol. 21, no. 5, pp. 475–480, 1995.
- [27] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.
- [28] A. Burns, "Preemptive priority based scheduling: an appropriate engineering approach," Tech. Rep. 214, University of York, 1994.
- [29] P. Simons, I. Niemelä, and T. Soeninen, "Extending and implementing the stable model semantics," *Artificial Intelligence*, vol. 138, no. 1–2, pp. 181–234, 2002.
- [30] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele, "A user's guide to gringo, clasp, clingo, and iclingo," November 2008.
- [31] N. Een and N. Sorensson, "Translating pseudo-boolean constraints into sat," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 2, pp. 1–26, 2006.
- [32] <http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html>.
- [33] T. M. Schmidl and D. C. Cox, "Robust frequency and timing synchronization for OFDM," *IEEE Transactions on Communications*, vol. 45, no. 12, pp. 1613–1621, 1997.
- [34] J. Heiskala and J. Terry, *OFDM Wireless LANs: A Theoretical and Practical Guide*, SAMS Publishing, Indianapolis, Ind, USA, 2002.
- [35] R. V. Nee and R. Prasad, *OFDM for Wireless Multimedia Communications*, Artech House, Boston, Mass, USA, 2000.
- [36] C.-F. Hsu, Y.-H. Huang, and T.-D. Chiueh, "Design of an OFDM receiver for high-speed wireless LAN," in *Proceedings of IEEE International Symposium on Circuits and Systems*, vol. 4, pp. 558–561, Sydney, Australia, May 2001.
- [37] F. Adachi, M. Sawahashi, and H. Suda, "Wideband DS-CDMA for next-generation mobile communications systems," *IEEE Communications Magazine*, vol. 36, no. 9, pp. 56–69, 1998.
- [38] H. Schulze and C. Lüders, *Theory and Application of OFDM and CDMA*, John Wiley & Sons, New York, NY, USA, 2005.
- [39] K. Kettunen, "Enhanced maximal ratio combining scheme for RAKE receivers in WCDMA mobile terminals," *Electronics Letters*, vol. 37, no. 8, pp. 522–524, 2001.
- [40] O. Atak, A. Atalar, E. Arikan, et al., "Design of application specific processors for the cached FFT algorithm," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '06)*, vol. 3, pp. 1028–1031, Toulouse, France, May 2006.
- [41] <http://lpsolve.sourceforge.net/5.5>.