Improving System Level Design Space Exploration by Incorporating SAT-Solvers into Multi-Objective Evolutionary Algorithms

Thomas Schlichter*, Martin Lukasiewycz, Christian Haubelt, and Jürgen Teich
Department of Computer Science 12
University of Erlangen-Nuremberg, Germany
{schlichter, haubelt, teich}@cs.fau.de

Abstract

Automatic design space exploration at the system level is the task of finding optimal or close to optimal mappings for a set of applications onto an optimized architecture. Especially, finding a feasible binding of processes onto resources that permit the communications imposed by data dependencies is known to be a $\mathcal{NP}$-complete task which demands the use of heuristic optimization approaches. Nearly all optimization approaches known from literature will fail in design spaces containing only a few feasible solutions. In this paper, we propose a novel approach based on the combination of Multi-Objective Evolutionary Algorithms and SAT-solvers to overcome these drawbacks. We will provide experimental results showing the efficiency of our novel methodology for synthetic and real life test cases.

1. Introduction

Modern embedded systems often consist of many communicating processor cores. The challenge in designing such heterogeneous multi-processor systems is to find optimal implementations with regard to multiple objectives while meeting several constraints. In order to allow an unbiased search, the task of *design space exploration* is performed before selecting (*decision making*) the actual implementation. Design space exploration is a very challenging constrained multi-objective optimization task [3]. The basic problem is the selection of appropriate hardware resources and the assignment of processes to the selected resources.

Due to data dependencies among the processes, nearly all possible implementations may be *infeasible*. Finding a feasible solution is known to be an $\mathcal{NP}$-complete task [3]. Thus, many researchers propose the use of Multi-Objective Evolutionary Algorithms (MOEAs) to solve these problems [8, 7]. In former work, we have proposed a method that integrates symbolic techniques using BDDs into MOEAs, that guide the search towards the feasible region [11]. Nevertheless, BDDs cannot be used to solve the complete problem of always finding a feasible solution if there is one at all, as the size of a BDD grows exponentially with the number of variables.

In this paper we will present an even more sophisticated decoding technique based on another symbolic method known from the formal verification area: SAT-solver [1]. Typical for SAT-solvers, this approach is not memory limited, and for our test cases we could also see that the runtime is comparable to other methods.

The remaining of this paper is structured as follows: Section 2 contains the definition of the search space and the task of design space exploration. In Section 3, we describe how this optimization problem can be solved using Multi-Objective Evolutionary Algorithms. Section 4 compares a straight forward decoding technique with the new one based on a SAT-solver. We will provide experimental results showing the advantages of the SAT decoding technique in Section 5. Finally, Section 6 concludes the paper.

2. Problem Statement

In this paper, we consider the problem of design space exploration for embedded systems. Basically, the design space exploration problem is a constrained multi-objective selection and assignment problem.

To allow for a mathematical model of the search space, the concept of a so-called *specification graph* is needed. A specification graph specifies a multi-processor system by means of its applications, the possible architecture, and the relation between these two views. Here, we use a graph-based approach already proposed by Blickle et al. [3].

Definition 1 (Specification Graph [3]) *A* specification graph *is a directed graph $g_s(V_s, E_s)$ that consists of a process graph $g_p(V_p, E_p)$, an architecture graph $g_a(V_a, E_a)$, and a set of* mapping edges E_m. *In particular, $V_s = V_p \cup V_a$, $E_s = E_p \cup E_a \cup E_m$, where $E_m \subseteq V_p \times V_a$.*

*supported by the Fraunhofer IIS, Germany

Proceedings of the 2006 Emerging VLSI Technologies and Architectures (ISVLSI'06)
0-7695-2533-4/06 $20.00 © 2006 IEEE

Consequently, mapping edges relate the vertices of the process graph to vertices of the architecture graph. The edges represent user-defined mapping constraints in the form of a relation: "can be implemented by".

The goal of design space exploration is to find optimal solutions which satisfy the specification given by the specification graph. Such a solution is called a *feasible implementation* of the embedded systems.

Our optimization algorithm uses evolutionary algorithms to find optimal implementations. Usually there are several conflicting optimization goals (e.g. speed vs. energy consumption), thus there are many optimal implementations.

An implementation, consists of three parts: (1) the *allocation* that indicates which elements of the architecture graph are used in the implementation, (2) the *binding*, i.e., the set of mapping edges which define the binding of processes to resources of the architecture graph, and (3) the *schedule* assigning a start time to each operation in the process graph.

Before defining the term *implementation* formally, Blickle et al. [3] introduce the so-called *activation* of vertices and edges:

Definition 2 (Activation [3]) *The* activation *of a specification graph $g_s(V_s, E_s)$ is a function* $a : V_s \times E_s \mapsto \{0,1\}$ *that assigns to each edge $e \in E_s$ and to each vertex $v \in V_s$ the value* 1 *(activated) or* 0 *(not activated).*

For the sake of simplicity, it is assumed that all vertices $v \in V_p$ and all edges $e \in E_p$ of the process graph g_p are activated subsequently. So only the vertices $v \in V_a$ of the architecture graph and the edges $e \in E_a \cup E_m$ can be either activated or deactivated.

An *allocation* α of a given specification graph g_s is the subset of all activated vertices and edges of the architecture graph g_a, i.e., $\alpha = \{v \in V_a \mid a(v) = 1\} \cup \{e \in E_a \mid a(e) = 1\}$. A *binding* β of a given specification graph g_s is the subset of activated mapping edges E_m, i.e., $\beta = \{e \in E_m \mid a(e) = 1\}$.

In order to restrict the search space, it is useful to determine the set of *feasible allocations* and *feasible bindings*. A feasible binding guarantees that communications demanded by the process graph can be established in the allocated architecture. This property makes the resulting optimization problem $\mathcal{NP}$-complete [3].

Definition 3 (Feasible Binding) *Given a specification graph g_s and an allocation α, a* feasible binding *is a binding β that satisfies the following requirements:*

1. *Each activated mapping edge $e \in \beta$ ends at an activated vertex, i.e., $\forall e = (v_p, v_a) \in \beta : v_a \in \alpha$.*
2. *For each process graph vertex $v_p \in V_p$, exactly one outgoing mapping edge $e \in E_m$ is activated, i.e.,*
$$\left|\left\{e \in \beta \mid e = (v_p, v_a), v_a \in V_a\right\}\right| = 1.$$
3. *For each process graph edge $e \in (v_i, v_j) \in E_p$:*
   - *either both operations are mapped onto the same vertex, i.e., $\widetilde{v}_i = \widetilde{v}_j$ with $(v_i, \widetilde{v}_i), (v_j, \widetilde{v}_j) \in \beta$,*
   - *or there exists an activated edge $\widetilde{e} = (\widetilde{v}_i, \widetilde{v}_j) \in E_a \cap \alpha$ in the architecture graph to handle the communication associated with edge e, i.e.,*
$$(\widetilde{v}_i, \widetilde{v}_j) \in E_a \cap \alpha \text{ with } (v_i, \widetilde{v}_i), (v_j, \widetilde{v}_j) \in \beta.$$

The term *feasible allocation* is used to indicate an allocation α that allows at least one feasible binding β.

Definition 4 (Implementation) *Given a specification graph g_s, a* (feasible) implementation ψ *is a triple (α, β, τ) where α is a feasible allocation, β is a corresponding feasible binding, and τ is a schedule.*

Now, the task of system synthesis can be formulated as a combinatorial *Multi-objective Optimization Problem*.

Definition 5 (System Synthesis) *The task of* system synthesis *is the following multi-objective optimization problem (MOP) where without loss of generality, only minimization problems are assumed here:*

minimize $f_1(x), f_2(x), \ldots, f_n(x)$,
subject to:
x represents a feasible implementation ψ

where $f(x) = (f_1(x), f_2(x), \ldots, f_n(x)) \in Y$ is the objective function*, Y is the* objective space*, $x = (x_1, x_2, \ldots, x_m) \in X$ is the* decision vector *and X is the* decision space.

3. Design Space Exploration

In this section, we will summarize how to solve the system synthesis problem by using MOEAs [5]. As proposed in [3], the MOEA determines the optimal allocation and bindings with respect to different objective functions. Afterwards, the schedule of an implementation is computed by a list scheduler.

The specification graph consisting of the process graph, the architecture graph, and a set of mapping constraints is used by an MOEA to determine the optimal implementations. Due to several even conflicting objective functions, multi-objective optimization problems generally do not contain only one global optimum, but a set of so-called *Pareto points*. A Pareto-optimal implementation is a design which is not worse than any other feasible solution in the design space in all objectives.

The MOEA requires a meaningful *encoding* for an implementation that consists of an allocation and a binding. Obviously, if allocations and bindings may be randomly

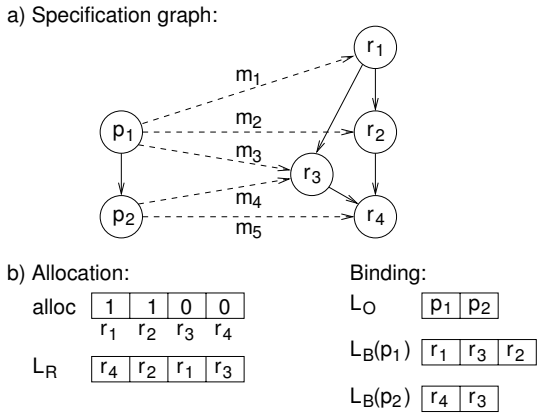

Figure 1. a) An example specification graph g_S with b) an individual consisting of the allocation bitmap *alloc* and different priority lists.

chosen, a lot of them can be infeasible. Here, we use a repairing strategy to partially repair infeasible solutions.

The allocation of resources is directly encoded in the so-called *individual*, which contains the structures shown in Figure 1 b). The bit vector *alloc* encodes for each resource $v_a \in V_a$ if it is activated or not. As this simple encoding may result in many infeasible allocations, there is a ordered *repair allocation list* L_R, that contains all resources $v_a \in V_a$ and allows to repair the allocation. A first simple heuristic only adds new resources $v_a \in V_a$ to the allocation and reflects the simplest case of infeasibility that may arise from non-executable processes $v_p \in V_p$: Consider the set $V_B \subseteq V_p$ that contains all processes that can not be executed, because not a single corresponding resource vertex is allocated, i.e., $V_B = \{v \in V_p \mid \forall (v,\widetilde{v}) \in E_m : a(\widetilde{v}) = 0\}$. To make the allocation feasible (in this sense), we add for each $v \in V_B$ one $\widetilde{v} \in V_a$, until feasibility in the sense above is achieved.

Besides the allocation, also the binding is encoded in the individual. Therefore the *binding order list* L_O is used, which indicates in which order to bind the processes $v_p \in V_p$. For each of these processes the individual also contains a *binding priority list* L_B, which contains the resources connected to this process by the mapping edges $e \in E_m$. These priority lists allow to decode every individual to a feasible implementation, but this decoding is a hard task.

In this paper, this decoding of an individual is of special interest and will be discussed in more detail in the next section.

4. Integrating SAT-Solvers into the MOEA

Former methods used to determine a feasible implementation often fail in design spaces with just a few feasible bindings, because of their local view on the specification.

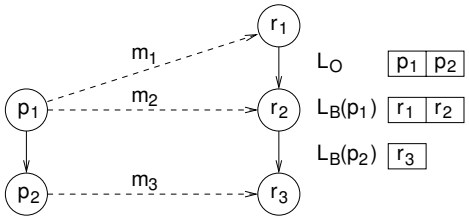

Figure 2. The sequential feasibility checker does not find a solution with the given binding order list L_O and binding priority lists L_B.

In the following, we will show that a) using a SAT-solver allows to have a global view on the specification and therefore guarantees to always find a feasible binding for a given allocation. b) Furthermore, if there is no feasible binding for a given allocation, the SAT-solver allocates additional resources to reach a feasible implementation.

The following subsections show the functionality of a simple heuristic called *sequential feasibility checker* that is used by existing design space exploration tools and will be used for comparison in Section 5. Here, we also present a symbolic representation of the complete specification and how a SAT-solver finds a feasible implementation using the information encoded in the individual.

4.1. Sequential Feasibility Checker

In this fairly simple constructive feasibility checker, each process is bound in the order it appears in the binding order list L_O of the individual. When binding a process, the different mapping edges are tested in the order of the binding priority list L_B. If the tested mapping edge is feasible with the binding performed up to this test, it is used.

So, which mapping edges are activated strongly depends on both the binding order list L_O and the binding priority list L_B. As only the yet bound processes are considered, the feasibility checker might be trapped, even if the allocation is feasible. An example is shown in Figure 2. Using the given binding order list L_O and the given binding priority lists L_B, the sequential feasibility checker does not find the valid binding $\beta = \{(p_1, r_2), (p_2, r_3)\}$. This happens because p_1 is first bound to r_1 what does not conflict with any already activated mapping edge (as there is none of that kind). However, that binding prohibits a feasible mapping of process p_2. As the sequential feasibility checker does not modify a chosen binding, it fails to find the valid binding.

4.2. Symbolic Representation of the Specification

The problem of finding a feasible implementation can be transformed into a satisfiability problem [6, 11, 10] by assigning Boolean variables to mapping edges and resources.

These Boolean variables indicate if the associated element is activated. In order to apply a common SAT-solver, the function has to be given in conjunctive normal form (CNF). This can be done by the following interpretation of the three requirements from Definition 3:

The first requirement states that each activated mapping edge $m = (p,r)$ has to end at an activated resource r:

$$\bigwedge_{m=(p,r)\in E_\mathrm{m}} \overline{\mathrm{a}(m)} \vee \mathrm{a}(r) \tag{1}$$

The second requirement states that exactly one mapping edge is activated for each process p, what can be split up into two statements. At least one mapping edge m has to be activated for each process:

$$\bigwedge_{p\in V_\mathrm{p}} \bigvee_{m=(p,r)\in E_\mathrm{m}} \mathrm{a}(m) \tag{2}$$

And at most one mapping edge has to be activated for each process.

$$\bigwedge_{m_i=(p,r_i),m_j=(p,r_j)\in E_\mathrm{m}:r_i\neq r_j} \overline{\mathrm{a}(m_i)} \vee \overline{\mathrm{a}(m_j)} \tag{3}$$

The third and last requirement states that communicating processes have to be mapped to the same or to an adjacent resource. This can be expressed by the following equation:

$$\bigwedge_{\substack{m_i=(p_i,r_i)\in E_\mathrm{m}:\\ (p_i,p_j)\in E_\mathrm{p}}} \left[\overline{\mathrm{a}(m_i)} \bigvee_{\substack{m_j=(p_j,r_j)\in E_\mathrm{m}:\\ r_i=r_j\vee(r_i,r_j)\in E_\mathrm{r}}} \mathrm{a}(m_j) \right] \tag{4}$$

There exists a feasible implementation iff the conjunction of Equation (1) - (4) is satisfiable. A SAT-solver can be used to determine a variable assignment representing a feasible solution. In the following section we will present a decoding algorithm using a SAT-solver. The advantage of this SAT-solver decoding technique is that it guarantees to find a feasible solution if such exists.

4.3. Davis-Putnam Decision Strategy Adjustment

To be able to decode an individual, the SAT-solver must regard the encoded information. Here we describe how to manage this by using a complete SAT-solver based on the Davis-Putnam (DP) backtrack search algorithm [4].

First, a decision strategy selects an unassigned variable which will be set to a fixed Boolean value. If the resulting CNF is recognized as unsatisfiable, the SAT-solver tries to resolve this conflict by a backtracking procedure. This continues until all variables are assigned, and this assignment represents a feasible solution. For our system-synthesis problem we developed a problem-specific decision strategy that processes the encoded information of the individual in the following three steps:

1. The Boolean value 1 is assigned to all allocated resources. The order does not matter as an allocation of a resource does not prohibit a feasible binding.

2. The Boolean value 0 is assigned to every not activated resource in the reverse order of the allocation priority list L_R. This ensures the sequential addition of resources respecting the priority in L_R if the CNF is unsatisfiable with the given resource allocation.

3. The SAT-solver tries to find a feasible binding based on the allocation determined in step 1 and step 2. The Boolean variables representing mapping edges are set to 1 regarding the order of the binding order list L_O and the corresponding binding priority list L_B. If this assignment prohibits a feasible binding, the next variable in the binding priority list is tested. If the end of the binding priority list is reached without finding a feasible assignment, the backtracking algorithm will automatically return to step 2 and add additional resources.

5. Experimental Results

In this section, we present experimental results from using a) the sequential feasibility checker respectively b) the SAT decoding technique. The PISA (Platform independent Interface for Search Algorithms [2]) framework was chosen for optimization purposes. In the present work, the SPEA2 selection procedure [12] was applied. For the SAT decoding technique, we used the zChaff [9] SAT-solver.

5.1. Problem Instances and Parameters

We created nine different classes of MOP instances (specification graphs). First we created three classes that differ in size. These are generated from following parameters: (i) The number of available resources in the architecture graph is 25, 50, respectively 75. (ii) The number of processes in the process graph g_p is either 50, 100, or 150. (iii) Each process has $3\ldots6, 4\ldots8$, respectively $5\ldots9$ random mapping edges. (iv) The number of edges in the process graph is given by a probability value. This value determines the probability that two processes are connected by an edge and is 25% for every problem class. All these values are typical in system level design.

Then for each of these three problem classes we created three different subclasses with feasibility probabilities 15%, 25%, and 45%. This probability is used when the edges E_a of the architecture graph are created. For each possible mapping edge $m_i = (p_i,r_i), m_j = (p_j,r_j) \in E_\mathrm{m}$ of two adjacent processes p_i and p_j, the resources r_i and r_j are connected with this probability to satisfy the data dependency.

Smaller probability values result in less created edges, and less feasible solutions exist.

For each of these nine problem classes we created 10 different problem instances. The genetic algorithm was run 10 times for each problem instance with both techniques.

We chose the parameters for the MOEA as follows: The population size was set to 100. For recombination, 25 children were created from 25 parents by single-point crossover. The mutation rate was set to $|decision\ variables|^{-1}$. The mutation operation is either single bit flip or order-based mutation.

5.2. Quantitative Results

To allow a comparison between the different optimization runs X_a, we combined all non-dominated points of a problem instance into a reference set X_R. This comparison was done using the ε-*dominance* performance indicator which scales the normalized points of the reference set X_R with an ε-value until this set does not dominate the particular set X_a anymore. So, smaller ε-values are better, and an ε-value of 1 means that the optimization run exactly reached the reference set. A detailed discussion on performance indicators can be found in [13].

Each optimization run is 1000 generations, and for every generation the average time to reach this generation is calculated and used to draw Figure 3. For each of the nine different problem classes, the average ε-values are calculated from the 100 different optimization runs. An optimization run only contributes to the average values after it found at least one valid solution.

Figure 3 a) shows, that the sequential feasibility checker is competitive to the SAT decoding technique for problems with a small number of processes and many feasible solutions. The less feasible solutions exist, the worse performs the sequential feasibility checker compared to the SAT decoding technique.

In Figure 3 b) one can see that the sequential feasibility checker becomes worse for more processes. The spikes in the curve for the sequential feasibility checker with a feasibility of 15% are due to optimization runs that find the first feasible solutions quite late.

Figure 3 c) indicates that the SAT decoding technique even performs well for a large number of processes compared to the sequential feasibility checker. The curve of the sequential feasibility checker with a feasibility of 15% shows that only a single run out of 100 was able to find any solutions at all.

If we compare the times for the different optimization runs, we can see that the runs of both decoding techniques take a longer time for bigger problems. This is due to the increased number of mapping edges, processes and resources which increase the runtime of the different quality functions. We can also see that the SAT-solver requires more time to compute 1000 generations than the sequential feasibility checker. This has two reasons:

1. The SAT decoding technique requires slightly more time to compute a binding for each individual (but one has to keep in mind that it always finds a feasible binding by adjusting the resource allocation properly).

2. As the SAT decoding technique finds more feasible solutions in each generation, the quality functions have to be executed for this increased number of individuals.

But even with these increased execution times the SAT decoding technique generates better solutions earlier for big problems that contain few feasible solutions. 150 processes is not the upper limit for the SAT decoding technique, internal test have shown that even 500 processes can be handled.

5.3. Real Life Example

Besides these synthetic test cases we also used the SAT decoding technique to optimize a real life example. This example describes an adaptive light control system of modern cars which has shown to be a very hard optimization task. The system is modeled by 234 communicating processes which can be mapped onto 1103 resources via 1851 overall mapping edges.

We tested the sequential feasibility checker and the SAT decoding technique to optimize this system. Due to space limitations, we omit exact figures and discuss the most important results only. If the sequential feasibility checker is used with a random initial population, the MOEA requires about 100 generations to find the first feasible solutions. Even if all resources are activated for all individuals of the initial population, the MOEA requires about 14 generations to find feasible solutions. The SAT decoding technique provides feasible solutions always beginning with the first generation. And even for later generations, our tests have shown that the SAT decoding technique always generated significant better solutions than the sequential feasibility checker after the same number of generations and even after the same optimization time.

6. Conclusions

In this paper, we have shown how to integrate SAT-solvers into Multi-Objective Evolutionary Algorithms in order to improve the convergence of the multi-objective optimization. We have presented experimental results from the area of automatic design space exploration of embedded systems which show the efficency of SAT-solvers in the necessary implementation decoding. From these experiments,

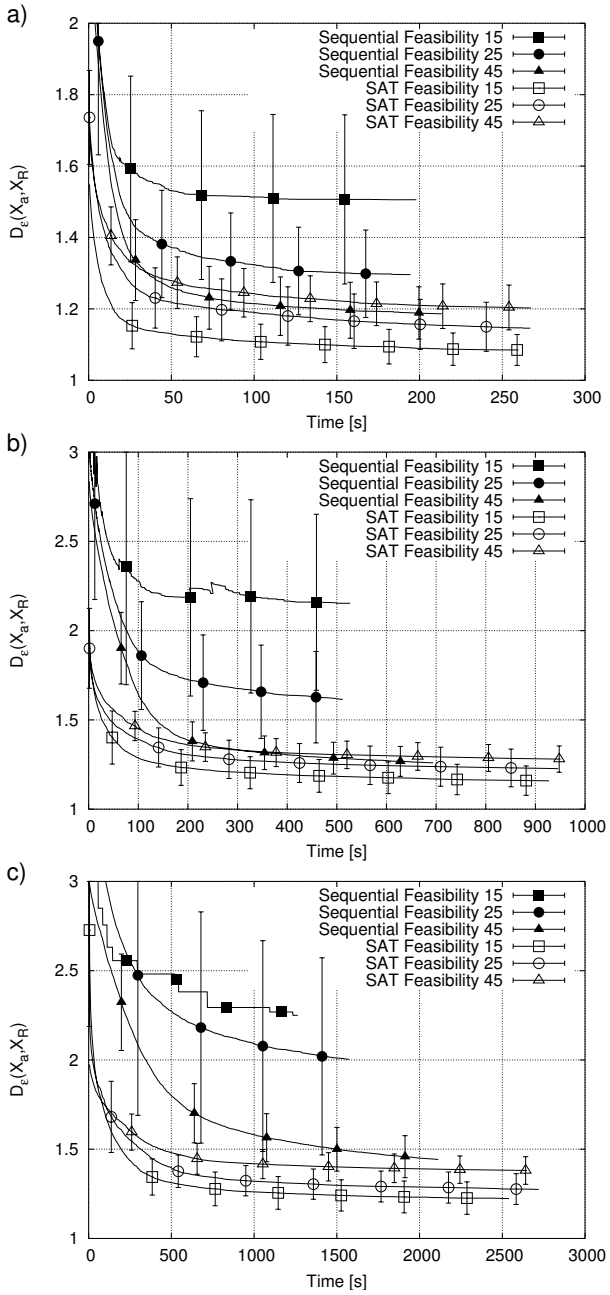

Figure 3. The mean ε-dominance over the average time for a)50, b)100, c)150 processes. The vertical bars indicate the standard deviation.

we conclude that our proposed method is especially well suited for problems with a search space containing many infeasible solutions.

Although the focus in this paper is on automatic design space exploration, there is the potential to generalize our results to other constrained combinatorial optimization problems. This issue will be investigated in future work.

References

[1] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. *Lecture Notes in Computer Science*, 1579:193–207, 1999.

[2] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA - A Platform and Programming Language Independent Interface for Search Algorithms. In *Lecture Notes in Computer Science (LNCS)*, volume 2632, pages 494–508, Faro, Portugal, Apr. 2003.

[3] T. Blickle, J. Teich, and L. Thiele. System-Level Synthesis Using Evolutionary Algorithms. In R. Gupta, editor, *Design Automation for Embedded Systems*, 3, pages 23–62. Kluwer Academic Publishers, Boston, Jan. 1998.

[4] M. Davis and H. Putnam. A computing procedure for quantification theory. *J. Assoc. Comput. Mach.*, 7:201–215, 1960.

[5] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, Ltd., Chichester, New York, Weinheim, Brisbane, Singapore, Toronto, 2001.

[6] R. Feldmann, C. Haubelt, B. Monien, and J. Teich. Fault Tolerance Analysis of Distributed Reconfigurable Systems Using SAT-Based Techniques. In P. Y. K. Cheung, G. A. Constantinides, and J. T. de Sousa, editors, *Field-Programmable Logic and Applications, Lecture Notes in Computer Science (LNCS)*, volume 2778, pages 478–487, Berlin, Heidelberg, Sept. 2003. Springer.

[7] C. Haubelt, S. Mostaghim, F. Slomka, J. Teich, and A. Tyagi. Hierarchical Synthesis of Embedded Systems Using Evolutionary Algorithms. In R. Drechsler and N. Drechsler, editors, *Evolutionary Algorithms for Embedded System Design*, Genetic Algorithms and Evolutionary Computation (GENA), pages 63–104. Kluwer Academic Publishers, Boston, Dordrecht, London, 2003.

[8] V. Kianzad and S. S. Bhattacharyya. CHARMED: A Multi-Objective Co-Synthesis Framework for Multi-Mode Embedded Systems. In *Proceedings of the 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP'04)*, pages 28–40, Galveston, U.S.A., Sept. 2004.

[9] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, 2001.

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

[11] T. Schlichter, C. Haubelt, F. Hannig, and J. Teich. Using Symbolic Feasibility Tests during Design Space Exploration of Heterogeneous Multi-Processor Systems. In *Proceedings of Application-specific Systems, Architectures and Processors (ASAP)*, Samos, Greece, July 2005.

[12] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In *Evolutionary Methods for Design, Optimisation, and Control*, pages 19–26, Barcelona, Spain, 2002.

[13] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation*, 7(2):117—132, Apr. 2003.