In *Field-Programmable Logic and Applications*
by Peter Y. K. Cheung, George A. Constantinides, and Jose T. de Sousa (Eds.).
In Lecture Notes in Computer Science (LNCS), Volume 2778, © Springer, Berlin, Heidelberg, 2003.

Fault Tolerance Analysis of Distributed Reconfigurable Systems Using SAT-Based Techniques*

Rainer Feldmann[1], Christian Haubelt[2], Burkhard Monien[1], and Jürgen Teich[2]

[1] AG Monien, Faculty of CS, EE, and Mathematics,
University of Paderborn, Germany
{obelix, bm}@upb.de
[2] Department of Computer Science 12, Hardware-Software-Co-Design
University of Erlangen-Nuremberg, Germany
{haubelt, teich}@cs.fau.de

Abstract The ability to migrate tasks from one reconfigurable node to another improves the fault tolerance of distributed reconfigurable systems. The degree of fault tolerance is inherent to the system and can be optimized during system design. Therefore, an efficient way of calculating the degree of fault tolerance is needed. This paper presents an approach based on satisfiability testing (SAT) which regards the question: How many resources may fail in a distributed reconfigurable system without losing any functionality? We will show by experiment that our new approach can easily be applied to systems of reasonable size as we will find in the future in the field of body area networks and ambient intelligence.

1 Introduction

Distributed reconfigurable systems [1,2] are becoming more and more important for applications in the area of automotive, body area networks, ambient intelligence, etc. The most outstanding property of these systems is the ability of reconfiguration. In terms of system synthesis, this means that the binding of tasks to resources is not static, i.e., the binding changes over time. Recent research was focused on the OS support for FPGAs [3] by dynamically assigning hardware tasks to an FPGA.

In a network of connected FPGAs it is possible to migrate hardware tasks from one node to another. Thus, resource faults can be compensated by *rebinding* tasks to fully functional nodes of the network. The process of rebinding is also called *repartitioning*. Distributed reconfigurable systems that support repartitioning possess an inherent fault tolerance. The degree of fault tolerance is a static property of the system and ,hence, can be optimized during system design. In order to evaluate the degree of fault tolerance, we define a new objective called *k-bindability*. A system is called *k-bindable* iff any set of k resources is redundant. Note, it may be possible that more than k resources are redundant but the k-bindability determines that k such that any set of k arbitrary resources can be removed from the system without losing any functionality.

* Supported in part by the German Science Foundation (DFG), SFB 376 (Massive Parallelität) and SPP 1148 (Rekonfigurierbare Rechensysteme).

The main contribution of this paper is to provide an efficient way based on SAT techniques to determine the k-bindability during system design. This problem is twofold: In a first step, we will reduce the well known binding problem from system synthesis to the satisfiability problem for boolean formulas. Next, we show how to calculate the k-bindability of a system using quantified boolean formulas (QBFs). Therefore, we focus on two particular system synthesis problems:

1. Does there exist a feasible binding for a given specification of a distributed reconfigurable system that supports repartitioning?
2. How many resources may fail in a distributed reconfigurable system that supports repartitioning without losing any functionality?

With this novel approach, we can optimize the fault tolerance of distributed reconfigurable system in an early design phase. In other words, we can maximize the k-bindability of such a system for a limited number of reconfigurable nodes and connections during design space exploration.

The problem to decide the satisfiability of QBFs is an important research issue in Artificial Intelligence, since QBF is the prototypical PSPACE-complete problem. Other PSPACE-hard problems from, e.g., conditional planning [4], non monotonic reasoning [5], and hardware verification [6] have been polynomially reduced to QBF. In the past several decision procedures for QBFs have been proposed in the literature [7,8,9,10].

This paper is structured as follows: In Section 2 we introduce the formal specification model of distributed reconfigurable systems used in this paper. The following section shows how to reduce the binding problem to the satisfiability problem of boolean formulas. In Section 4 a QBF-based approach to determine the k-bindability of a distributed reconfigurable systems that supports repartitioning is proposed. Finally, we will show by experiment (Section 5) that problem instances of reasonable size are easily solved by the Davis-Putnam based QBF solver QSOLVE [9].

2 Preliminaries

In order to specify distributed reconfigurable systems, we use a graph-based approach. First, we model the behavior of a system using a directed graph, called *task graph*. The vertices of the task graph represent tasks $t \in T$ where T is a finite set. The edges of the task graph model data dependencies $d \in D$ between the tasks, i.e., $D \subseteq T \times T$.

On the other hand, we model the architecture of our distributed reconfigurable system by a so-called *architecture graph*. An architecture graph is also a directed graph, where vertices correspond to reconfigurable nodes $r \in R$ of the network. Edges of the architecture graph model directed connections $c \in C \subseteq R \times R$ between the nodes.

To relate tasks $t \in T$ and reconfigurable nodes $r \in R$, mapping edges $m \in M$ map tasks to nodes. A mapping edge $m = (t, r)$ indicates that t may be executed on r. Note that more than one mapping edge could be associated with a task t or a reconfigurable node r, modeling possible bindings and resource sharing, respectively. Such graph-based models are also used in commercial systems like VCC [11].

Example 1. Figure 1 shows a specification of a distributed reconfigurable system. The set of tasks T and data dependencies D are given by $T = \{\mathrm{t}_0, \mathrm{t}_1, \mathrm{t}_2\}$ and $D =$

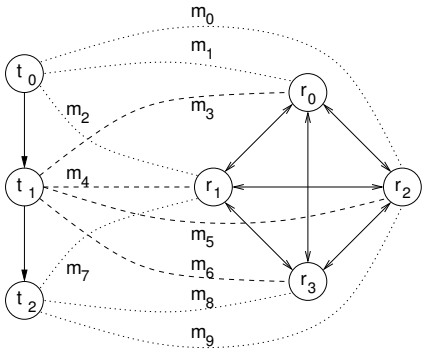

Figure 1. Distributed control system consisting of a sample task (t_0), a control task (t_1), and a driver task (t_2). The architecture is composed of four reconfigurable nodes ($\mathrm{r}_0, \ldots, \mathrm{r}_3$). The additional mapping edges ($\mathrm{m}_0, \ldots, \mathrm{m}_9$) describe possible bindings.

$\{(\mathrm{t}_0, \mathrm{t}_1), (\mathrm{t}_1, \mathrm{t}_2)\}$, respectively. This task graph models the coarse grain behavior of a distributed control system, where t_0 corresponds to a sample task sampling a sensor, t_1 corresponds to the control task implementing the actual control, and t_2 models the driver task driving an actuator.

The architecture graph in Figure 1 consists of the reconfigurable nodes $R = \{\mathrm{r}_0, \mathrm{r}_1, \mathrm{r}_2, \mathrm{r}_3\}$. Each reconfigurable node could directly communicate with each other, i.e., the architecture graph is a clique. The mapping edges $M = \{\mathrm{m}_0, \ldots, \mathrm{m}_9\}$ indicate that the sample task t_0 may be executed on any reconfigurable node r_0 to r_2 and the driver task t_2 may be performed on any of the reconfigurable node r_1 to r_3. The controller task t_1 could be bound to any of the reconfigurable nodes in the architecture graph.

3 Binding

With the model introduced previously, the task of system synthesis could be formulated as: "Find a feasible *binding* of the tasks $t \in T$ to reconfigurable nodes $r \in R$, i.e., a subset of mapping edges." Here, a binding is said to be feasible if:

1. each task $t \in T$ is bound to exactly one reconfigurable node $r \in R$ and
2. required communications given by the data dependencies $d \in D$ can be handled by the given architecture graph, i.e., if there is a directed edge $d = (t_i, t_j)$ between task t_i and task t_j then either t_i and t_j have to be performed on the same reconfigurable node r (intra-node communication) or on reconfigurable nodes r_i and r_j which are directly connected via an edge $c = (r_i, r_j)$ (inter-node communication).

Blickle et al. [12] have reduced the problem of finding a feasible binding to the boolean satisfiability problem which is NP-complete. In this paper, we show how to derive boolean functions from specifications given by a task graph, an architecture graph, and the mapping edges as described in Section 2 such that the boolean function is satisfiable iff the specified distributed reconfigurable system has a feasible binding. These function

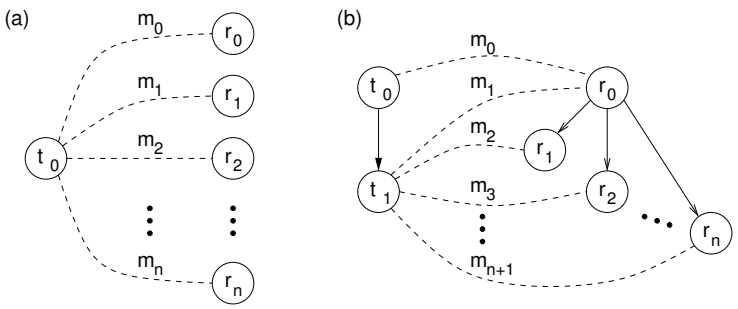

Figure 2. (a) For each task $\mathrm{t} \in \mathrm{T}$ exactly one outgoing mapping edge has to be activated. (b) In order to establish the required communication ($\mathrm{d} = (\mathrm{t}_0, \mathrm{t}_1)$), we have to execute the tasks t_0 and t_1 on the same reconfigurable node r_0 or on adjacent reconfigurable nodes.

could be tested by QBF solvers. Later, we extend this idea in order to analyze aspects such as whether a distributed reconfigurable system is fault tolerant and to what degree.

First, we consider the problem of checking the feasibility of a given binding. Therefore, we introduce some notations: Let m_i be a boolean variable, indicating if the mapping edge m_i is part of the binding ($m_i = 1$), or not ($m_i = 0$). The assignment of all variables m_i is denoted by (m). Note, that (m) is a binary coding of the binding. The set of all possible assignments of (m) is denoted by (M). With these new notations, we check the feasibility of a given binding represented by the coding (m) by solving a boolean equation. Therefore, we test both criteria of the feasibility as given above.

Example 2. First, we test if there is exactly one outgoing mapping edge for each task $t \in T$ in the binding. As an example consider Figure 2(a). There is a single task t_0 and $n + 1$ mapping edges $m_0, \ldots, m_n$. A boolean function that indicates if there is exactly one outgoing mapping edge for t_0 in conjunctive normal form (cnf) is: $(m_0 + m_1 + m_2 + \cdots + m_n) \cdot (\overline{m}_0 + \overline{m}_1)(\overline{m}_0 + \overline{m}_2) \ldots (\overline{m}_0 + \overline{m}_n) \cdot (\overline{m}_1 + \overline{m}_2) \ldots (\overline{m}_1 + \overline{m}_n) \ldots (\overline{m}_{n-1} + \overline{m}_n)$ Here, $+$ denotes the boolean OR and $\cdot$ is the boolean AND. The first clause ensures that at least one of the mapping edges is activated ($m_i = 1$). The remainder guarantees that at most one mapping edge is part of the binding. The conjunction of both parts results in the required property.

For each task $t \in T$ we have to establish a formula similar to the one given in Example 2. The logical product results in a boolean function $b_1 : (M) \rightarrow \{0, 1\}$ with $b_1((m)) = 1$ iff (m) contains exactly one mapping edge per task. Hence, we obtain Equation (1) where $\prod$ denotes the boolean AND and $\sum$ denotes the boolean OR.

$$b_1((m)) = \prod_{t \in T} \left[\left(\sum_{\substack{m \in M: \\ m=(t,r)}} m \right) \cdot \left(\prod_{\substack{m_i, m_j \in M: \\ m_i=(t,r_x) \wedge m_j=(t,r_y) \wedge r_x \neq r_y}} (\overline{m}_i + \overline{m}_j) \right) \right] \quad (1)$$

Now, that we are sure that the first criteria is fulfilled, we check the second property of feasible bindings. All data dependencies $d \in D$ must be provided by the architecture

of the implementation. Therefore, let $d = (t_i, t_j)$. If t_i is executed on r_x then t_j has to be performed on r_x also or on an adjacent reconfigurable node r_j, i.e., $(r_i, r_j) \in C$.

Example 3. Consider the example in Figure 2(b). The task t_0 is bound to r_0 by m_0. The execution of task t_1 must take place on node r_0 itself (by m_1) or on any adjacent reconfigurable node $r_1, \ldots, r_n$. An implication that assures this property is given by $m_0 \mapsto (m_1 + m_2 + \cdots + m_{n+1})$. This is equivalent to the clause: $(\overline{m}_0 + m_1 + m_2 + m_3 + \cdots + m_{n+1})$. This equation needs to be satisfied for each mapping edge $m \in M$.

A boolean function $b_2 : (M) \rightarrow \{0, 1\}$ to check the second property of feasibility is therefore:

$$b_2\left((m)\right) = \prod_{\substack{m=(t,r)\in M,\\ t_i\in T:(t,t_i)\in D}} \left[\overline{m} + \sum_{\substack{m_j\in M:m_j=(t_i,r_x)\wedge\\ (r=r_x\vee(r,r_x)\in C)}} m_j \right] \tag{2}$$

With Equation (1) and (2), we formulate a boolean function $b : (M) \rightarrow \{0, 1\}$ to check the feasibility of a given binding coded by (m):

$$b\left((m)\right) = b_1\left((m)\right) \cdot b_2\left((m)\right) \tag{3}$$

b is given in cnf and $b\left((m)\right) = 1$ iff the system has a feasible binding. In order to check if there is at least one feasible binding for a given specification, a SAT solver may be used to solve the following problem

$$\exists (m) : b\left((m)\right) \tag{4}$$

Checking whether a binding is feasible or whether a partial binding may be completed can be an important task during synthesis, but also in dynamically reconfigurable distributed systems. One application of the above SAT-techniques is therefore the domain of fault tolerance.

4 Fault Tolerance

In reconfigurable systems, the binding may change over time. Therefore, it may be possible to compensate resource faults by *rebinding* tasks to fully functional resources. The process of rebinding is called *repartitioning*. Recent research is focused on the OS support for single FPGA architectures [3]. In this section, we show how to model resource faults and how to measure the robustness of a given distributed reconfigurable system that supports repartitioning, also using SAT-based techniques. Therefore, we define the so-called k-*bindability* which quantifies the number of redundant resources in such a system.

4.1 Modeling Resource Faults

If a reconfigurable nodes fails, i.e., we cannot use this node for task execution any longer, the *allocation* of resources nodes may change. The allocation is the set of used

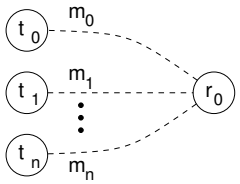

Figure 3. In case of a resource defect, all incoming mapping edges must be deactivated.

resources in our implementation. Furthermore, an allocation is said to be feasible if there exists at least one feasible binding for this allocation. As in the case of the binding, we use the term (r) as the coding of an allocation where $r_i = 1$ indicates that the reconfigurable node r_i is part of the allocation. Furthermore, the term (R) describes the set of all possible allocation codings. If a reconfigurable node r_i fails, we have to set the associated binary variable r_i to zero ($r_i = 0$). All adjacent mapping edges m_j to r_i become meaningless, and should not be used in the binding, i.e., $m_j = 0$.

Example 4. Figure 3 shows a single reconfigurable node r_0 and n mapping edges. If r_0 fails $\mathrm{m}_0, \ldots, \mathrm{m}_\mathrm{n}$ must not be used in the binding. We express this fact by n implications in the form $\overline{\mathrm{r}}_0 \mapsto \overline{\mathrm{m}}_j$ with $j = 0, \ldots, n$. In cnf we get: $(r_0 + \overline{m}_0)(r_0 + \overline{m}_1)(r_0 + \overline{m}_2) \ldots (r_0 + \overline{m}_n)$.

Again, we propose a boolean function to deactivate all mapping edges adjacent to a defect reconfigurable node. This boolean function $e : (M) \times (R) \rightarrow \{0, 1\}$ is satisfiable iff no reconfigurable nodes fail or there exists a feasible binding not using any of the mapping edges to the defect node.

$$e\left((m),(r)\right) = \prod_{r \in R, m \in M : m = (t,r)} (r + \overline{m}) \tag{5}$$

With this formula, we can check if a given reconfigurable node is redundant. For example, if we want to test if r_0 is redundant we solve the following SAT formula:

$$\exists (m),(r) : \overline{r}_0 \cdot e\left((m),(r)\right) \cdot b\left((m)\right)$$

4.2 k-Bindability

A frequent question is how many resources could fail in a distributed reconfigurable system that supports repartitioning without losing the desired functionality. Therefore, we define the number of nodes that may fail as *k-bindability*, i.e., k is the maximum number such that any set of k reconfigurable nodes is redundant. Note that we can remove any $n < k$ nodes of our distributed system without losing the specified functionality.

Example 5. Figure 1 shows the specification for a distributed control system. Let us remove one of the reconfigurable nodes $\mathrm{r}_0, \ldots, \mathrm{r}_3$ in Figure 1. Whatever node we choose, by rebinding the tasks we retain a running system, i.e., our system is at least 1-bindable. If we simultaneously remove any two of the reconfigurable nodes, again, our system remains working through rebinding the tasks. Now, our system is at least 2-bindable.

Let us check for 3-bindability. If we remove the reconfigurable nodes $\mathrm{r}_0, \mathrm{r}_1, \mathrm{r}_3$ (or they fail simultaneously), we could bind all tasks $\mathrm{t}_0, \mathrm{t}_1$, and t_2 to node r_2. But the system is not 3-bindable, since if the nodes $\mathrm{r}_0, \mathrm{r}_1$, and r_2 fail simultaneously, we can not find any reconfigurable node to bind task t_0 to. That is: the system is 2-bindable.

In order to check for k-bindability using SAT-based techniques, we formulate a boolean function which encodes all system errors with exactly k reconfigurable node defects:

$$f^{(k)} = (A) \times (R) \rightarrow \{0, 1\} \tag{6}$$

This function depends on the auxiliary variables a_i with $i = 0, \ldots, a_{|R|-1}$ where $|R|$ denotes the cardinality of R. If exactly k auxiliary variables are set to zero, we set the k corresponding allocation variables r_i to zero, otherwise all allocation variables may be set to one (i.e., no node fails).

Example 6. For the reconfigurable nodes in Figure 1, we encode the single resource defect of r_0 as: $a_3 a_2 a_1 \overline{a}_0$. The implication $(a_3 a_2 a_1 \overline{a}_0) \mapsto \overline{r}_0$ forces the allocation variable r_0 to zero. In cnf this corresponds to: $(\overline{a}_3 + \overline{a}_2 + \overline{a}_1 + a_0 + \overline{r}_0)$.

For all possible faults with exactly one single resource defect, we have to encode $|R|$ different cases (for each resource):

$$f^{(1)}\left((a),(r)\right) = \prod_{j=0}^{|R|-1} \left(\overline{r}_j + a_j + \sum_{i=0, i\neq j}^{|R|-1} \overline{a}_i \right)$$

$f^{(1)}\left((a),(r)\right)$ does not impose any constraints on (r) if more than one variable a_i is set to false. With this boolean function, we check if for all of these faults there is at least one feasible binding. This is done by the following quantified boolean formula:

$$\forall(a)\exists(r),(m) : f^{(1)}\left((a),(r)\right) \cdot e\left((m),(r)\right) \cdot b\left((m)\right)$$

We extend this approach by encoding all faults with exactly k resource defects. This is again an implication and can be written in cnf as:

$$f^{(k)}\left((a),(r)\right) = \prod_{i_1=0}^{|R|-1} \cdots \prod_{i_k=i_{k-1}}^{|R|-1} \prod_{l=1}^{k} \left(\overline{r}_{i_l} + \sum_{\substack{n=0 \\ n=i_1 \vee \cdots \vee n=i_k}}^{|R|-1} a_n + \sum_{\substack{n=0 \\ n\neq i_1 \wedge \cdots \wedge n\neq i_k}}^{|R|-1} \overline{a}_n \right)$$

Note again, that $f^{(k)}\left((a),(r)\right)$ does not impose any constraints on (r) if $p > k$ variables a_i are set to false.

Now, that we know how to code resource defects in a boolean function, we formulate the general form of the QBF solving the k-bindability problem:

$$\forall(a)\exists(r),(m) : f^{(k)}\left((a),(r)\right) \cdot e\left((m),(r)\right) \cdot b\left((m)\right) \tag{7}$$

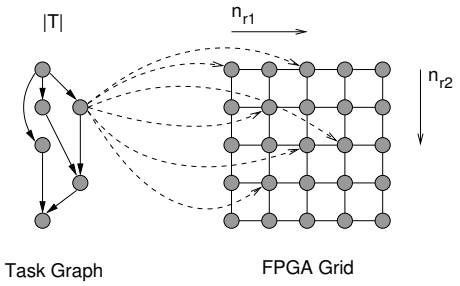

Figure 4. Example specification of an FPGA grid and an application consisting of $|T| = 6$ tasks. The grid is composed of $n_{\mathrm{r1}} \times n_{\mathrm{r2}} = 5 \times 5$ FPGAs. The number of mapping edges per task is given by $n_{\mathrm{m}} = 6$ and is shown only for one task.

5 Experimental Results

The k-bindability as defined above specifies the degree of fault tolerance of a distributed reconfigurable system. This fault tolerance can be optimized during system design. In order to compare different implementations during design space exploration, we must evaluate these implementations. In this section, we present first results of our new approach by using QSOLVE [9]. For this purpose, we design a benchmark. Our goal is to evaluate the runtime in dependence of the problem size (e.g., number of tasks, number of resources) which could be easily solved and, hence, could be investigated during automated design space exploration.

In a first step, an array of $n_{\mathrm{r1}} \times n_{\mathrm{r2}}$ reconfigurable nodes is defined. Communications are established such that the array is a 2-dimensional grid. Here, we use a grid in order to construct scalable architectures of distributed reconfigurable systems. Furthermore, a grid is typical for reconfigurable architectures as FPGAs and coarse grain architectures like PACT [13], Chameleon [14], etc. Next, we define a weakly connected random task graph with $|T| = n_{\mathrm{t}}$ tasks. The probability that there is a data dependency between task t_i and task t_j with $j > i$ is given by another parameter called pb. In a last step, n_{m} mapping edges are randomly drawn from each task $t \in T$ to reconfigurable nodes $r \in R$, i.e., there are $|M| = |T| \cdot n_{\mathrm{m}}$ mapping edges. Figure 4 shows an example of such a specification. The most meaningful results are presented in the following.

5.1 Feasibility of Binding

In a first test, we solve Equation (4) for randomly generated specifications. Here, three different $n_{\mathrm{r1}} \times n_{\mathrm{r2}}$ grids of reconfigurable nodes are investigated ($5 \times 5, 10 \times 10$, and 15×15). We map randomly generated task graphs onto each of these distributed architectures, while varying the number of tasks ($n_{\mathrm{t}} = 50, 100, 150$). The tasks are connected with a probability of $pb = 0.5$. The number of mapping edges is chosen in a way, that feasible as well as infeasible systems are constructed. Only the cases of infeasible bindings are documented here (Finding a feasible binding by Equation (4) is the

Table 1. Number of recursions recur, number of assignments assign, and computation time time required for solving (unsatisfiable) Equation (4).

	$\lvert T\rvert = 50$			$\lvert T\rvert = 100$			$\lvert T\rvert = 150$		
	5×5	10×10	15×15	5×5	10×10	15×15	5×5	10×10	15×15
n_{m}	17	65	140	19	75	160	20	80	–
$recur$	21	70	180	23	116	237	29	176	–
$assign$	2149	30285	136675	6527	120421	421960	12809	266205	–
$time/s$	0.07	1.80	14.05	0.46	14.33	69.67	1.57	43.11	–

Table 2. Number of recursions recur, number of assignments assign, and computation time time to solve the k-bindability equation (satisfiable) Equation (7).

$\lvert T\rvert$		$k=1$	$k=2$	$k=3$	$k=4$	$\lvert T\rvert$		$k=1$	$k=2$	$k=3$	$k=4$
	$recur$	532	2759	10230	30106		$recur$	587	4972	19136	49858
25	$assign$	8569	43856	163972	487057	40	$assign$	10179	77695	293135	827385
	$time/s$	0.10	0.60	2.22	6.72		$time/s$	0.23	1.78	7.06	20.26
	$recur$	434	3235	11931	35222		$recur$	649	4524	17107	51307
30	$assign$	7474	54488	201263	594378	45	$assign$	12109	86569	318774	933571
	$time/s$	0.12	0.94	3.62	10.69		$time/s$	0.35	2.47	9.15	26.28
	$recur$	488	3593	13403	39787		$recur$	633	4893	17818	53272
35	$assign$	9031	65010	240039	705878	50	$assign$	12573	93306	342441	1007054
	$time/s$	0.19	1.45	5.89	16.63		$time/s$	0.37	2.74	10.02	29.90

easier case). The average results (100 samples each) using QSOLVE [9] obtained on a PC system with a 1.8 GHz processor are shown in Table 1.

The number of recursions $recur$ corresponds to the number of nodes in the search tree. We see that systems with 225 reconfigurable nodes and 16,000 mapping edges be checked in a reasonable amount of time ($\approx$ 1 min). Note: We only construct weakly connected task graphs. If we were using n weakly connected task subgraphs that are not connected, our binding problem consists of n independent binding problems. A QBF solver solves these (sub)problems independent of each other, which is much easier.

5.2 k-Bindability

With the results above, we consider the k-bindability problem as described in Section 4. Table 2 shows the average results (100 samples each) obtained from solving the boolean functions with the QBF-solver QSOLVE. We have chosen a 4×4 grid of reconfigurable nodes. Different numbers n_{t} of tasks are mapped onto this architecture. With parameters $n_{\mathrm{m}} = 13$ and $pb = 0.5$, we check the k-bindability for $k = 4, \ldots, 1$. Table 2 shows that systems with 50 tasks are still solvable in a reasonable amount of time.

As mentioned above, our approach is not limited to grids of reconfigurable nodes but supports arbitrary topologies. Thus, it is possible to optimize the architecture of distributed reconfigurable systems by using SAT-based techniques during design space exploration.

6 Conclusions

Distributed reconfigurable systems, e.g., arrays of reconfigurable hardware elements including FPGAs or medium and coarse granular reconfigurable systems such as PACT [13] and Chameleon [14], possess an inherent fault tolerance which can be optimized during system design. The main contribution of this paper is to provide an efficient method to determine the degree of fault tolerance of a system, the so-called k-bindability. Two particular problems were considered in this paper: (i) Does there exist a feasible binding for a given specification of a distributed reconfigurable system that supports repartitioning? (ii) How many resources may fail in a distributed reconfigurable system that supports repartitioning without losing any functionality? Both problems were solved by reducing the binding problem to quantified boolean formulas and applying the QBF solver QSOLVE in order to test the satisfiability of these formulas. We have shown by experiment that our new approach can easily be applied to systems of reasonable size as we will find in the future in the field of body area networks and ambient intelligence. Hence, this approach provides a way to optimize the architecture of distributed reconfigurable systems during design space exploration.

References

1. Dick, R., Jha, N.: CORDS: Hardware-Software Co-Synthesis of Reconfigurable Real-Time Distributed Embedded Systems. In: Proceedigns of ICCAD'98. (1998) 62–68
2. Ouaiss, I., Govindarajan, S., Srinivasan, V., Kaul, M., Vemuri, R.: An Integrated Partitioning and Synthesis System for Dynamically Reconfigurable Multi-FPGA Architectures. In: IPPS/SPDP Workshops. (1998) 31–36
3. Walder, H., Platzner, M.: Online Scheduling for Block-partitioned Reconfigurable Devices. In: Proceedings of Design, Automation and Test in Europe (DATE03). (2003) 290–295
4. Rintanen, J.: Constructing Conditional Plans by a Theorem-Prover. Journal of Artificial Intelligence **10** (1999) 323–352
5. Egly, U., Eiter, T., Tompits, H., Woltran, S.: Solving Advanced Reasoning Tasks Using Quantified Boolean Formulas. In: Proc. of the 17th Nat. Conf. on Artificial Intelligence. (2000) 417–422
6. Scholl, C., Becker, B.: Checking Equivalence for Partial Implementations. In: Proceedings of 38th Design Automation Conference, Las Vegas, USA (2001) 238–243
7. Kleine-Büning, H., Karpinski, M., Flögel, A.: Resolution for Quantified Boolean Formulas. Information and Computation **117** (1995) 12–18
8. Cadoli, M., Giovanardi, A., Schaerf, M.: An Algorithm to Evaluate Quantified Boolean Formulae. In: Proc. of the 15th Nat. Conf. on Artificial Intelligence. (1998) 262–267
9. Feldmann, R., Monien, B., Schamberger, S.: A Distributed Algorithm to Evaluate Quantified Boolean Formulas. In: Proc. of the 17th Nat. Conf. on Artificial Intelligence. (2000) 285–290
10. Giunchiglia, E., Narizzano, M., Tacchella, A.: Backjumping for Quantified Boolean Formulas. In: Proc. of the 17th Int. Joint Conf. on Artificial Intelligence. (2001) 275–281
11. Cadence: Virtual Component Co-design (VCC). (2001) http://www.cadence.com.
12. Blickle, T., Teich, J., Thiele, L.: System-Level Synthesis Using Evolutionary Algorithms. In Gupta, R., ed.: Design Automation for Embedded Systems. 3. Kluwer Academic Publishers, Boston (1998) 23–62
13. Baumgarte, V., May, F., Nückel, A., Vorbach, M., Weinhardt, M.: PACT XPP - A Self-Reconfigurable Data Processing Architecture. In: ERSA, Las Vegas, Nevada (2001)
14. Chameleon Systems: CS2000 Reconfigurable Communications Processor. (2000)