SAT-Based Techniques in System Synthesis

Christian Haubelt, Jürgen Teich
Computer Science, University of Erlangen-Nuremberg

Rainer Feldmann, Burkhard Monien
AG-Monien, University of Paderborn

Abstract

In this paper, we show how to integrate SAT-based techniques into the task of system synthesis by regarding the the problems: (i) feasibility check and (ii) evaluation of quality.

1. Introduction

SAT-based verification of electronic systems became very popular in recent years [3]. In this paper, we show that SAT-techniques are also applicable and helpful during the synthesis and the optimization of a system. Therefore, we must consider two questions: (i) How to represent specifications and (ii) How to quantify properties of embedded systems by boolean formulas? Thus, we will reduce the well known binding problem to the boolean satisfiability problem. Next, we show how to quantify the degree of fault tolerance of a system using quantified boolean formulas (QBFs). These problems correspond to typical subroutines often used during design space exploration. We will show by experiment that problem instances of reasonable size are easily solved by the QBF solver QSOLVE [2].

2. Binding

To specify embedded systems, we use a graph-based approach. The behavior of a system is modeled by a so-called *process graph*. The vertices represent processes $p \in P$ and the directed edges are data dependencies $d \in D$ between processes. The architecture is modeled by a so-called *architecture graph*, where vertices correspond to resources $r \in R$ and edges are directed connections $c \in C$. Mapping edges $m \in M$ relate processes and resources, where $m = (p, r)$ indicates that p may be executed on r.

The task of system synthesis can be formulated as: "Find a feasible *binding* $\beta \subseteq M$ of the processes to resources." Blickle et al. [1] have reduced this problem to the boolean satisfiability problem. In this paper, we show how to derive boolean functions from these specifications such that the boolean functions are satisfiable iff the specified system has a feasible binding. Therefore, we need some notations:

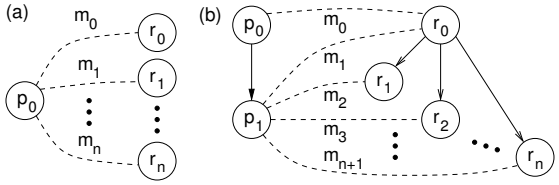

Figure 1. (a) For each process $\mathsf{p} \in \mathsf{P}$ exactly one outgoing mapping edge has to be activated. (b) To establish the communication $(\mathsf{p}_0, \mathsf{p}_1)$, we have to execute the processes p_0 and p_1 on the same or on adjacent resources.

Let m_i be a boolean variable, indicating if the mapping edge m_i is in the binding ($m_i = 1$), or not ($m_i = 0$). The assignment of all variables m_i is denoted by (m).

A binding is said to be feasible (see also [1]) if:

1. Each process is bound to exactly one resource (see also Figure 1(a)). This leads to the following boolean function which is satisfiable iff (m) contains exactly one mapping edge per process (Here, $+$ denotes the boolean OR and $\cdot$ is the boolean AND):

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

2. The data dependencies $d \in D$ can be handled by the resource graph, i.e., if there is an edge $d = (p_i, p_j)$ then either p_i and p_j have to be performed on the same resource r or on adjacent resources. Consider the example shown in Figure 1(b). An implication assuring this property is $(\overline{m}_0 + m_1 + m_2 + m_3 + \cdots + m_{n+1})$. This equation needs to be satisfied for each mapping edge:

$$b_2((m)) = \prod_{\substack{m=(p,r) \in M, \\ p_i \in P: \\ (p,p_i) \in D}} \left[\overline{m} + \sum_{\substack{m_j \in M: m_j=(p_i,r_x) \wedge \\ (r=r_x \vee (r,r_x) \in C)}} m_j \right] \quad (2)$$

We define the boolean function $b((m)) = b_1((m)) \cdot b_2((m))$ to test the feasibility of a binding, where $b((m)) = 1$ iff the system has a feasible binding. 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

$$\exists (m) : b((m)) \quad (3)$$

3. Fault Tolerance

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

Modeling Resource Faults If a resource fails, the *allocation* (set of used resources in an implementation) may change, where an allocation is said to be feasible if there exists at least one feasible binding for this allocation. As before, we use the term (r) as the coding of an allocation. If resource r_i fails, the binary variable r_i must be set to zero. All incident mapping edges m_j to resource r_i should not be used in the binding. We propose a boolean function to deactivate all incident mapping edges.

$$e\left((m),(r)\right) = \prod_{\substack{r \in R \\ m \in M : m=(p,r)}} (r + \overline{m}) \tag{4}$$

k-Bindability A frequent question is how many resources can fail without losing any functionality. We define this number as *k-bindability*, where k is the maximum number such that any set of k resources is redundant. To check this property using SAT-techniques, we formulate a boolean function $f^{(k)}$ which encodes all system errors with exactly k resource defects. This function depends on $|R|$ auxiliary variables. If exactly k auxiliary variables are zero, we set the corresponding allocation variables r_i to zero:

$$f^{(k)}\left((t),(r)\right) = \prod_{i_1=0}^{|R|-1} \prod_{i_2=i_1}^{|R|-1} \cdots \prod_{i_k=i_{k-1}}^{|R|-1} \cdot \prod_{l=1}^{k} \left(\overline{r}_{i_l} + \sum_{\substack{n=0 \\ n=i_1 \vee \cdots \vee n=i_k}}^{|R|-1} t_n + \sum_{\substack{n=0 \\ n \neq i_1 \wedge \cdots \wedge n \neq i_k}}^{|R|-1} \overline{t}_n \right)$$

$f^{(k)}$ does not impose any constraints on (r) if $p > k$ variables t_i are set to false. Now, the k-bindability problem is:

$$\forall(t) \exists(r),(m) : f^{(k)}\left((t),(r)\right) \cdot e\left((m),(r)\right) \cdot b\left((m)\right) \tag{5}$$

4. Experimental Results

To evaluate our new approaches, we design a benchmark: An $n_{\mathrm{r}1} \times n_{\mathrm{r}2}$ processor grid and a weakly connected random process graph with $|P|$ processes is defined. There is a data dependency between p_i and p_j with $j > i$ with a probability pb. Furthermore, n_{m} mapping edges are randomly drawn from each process to resources $r \in R$. The most meaningful results (PC with 1.8 GHz) of this benchmark are presented in the following.

Feasibility of Binding We solve Equation (3) for randomly generated specifications. The average results (100 samples) using QSOLVE [2] are shown in Table 1. Only the more difficult cases of infeasible bindings are documented here. The number of mapping edges is chosen such that feasible as well as infeasible systems are constructed.

Table 1. Results for testing (unsatisfiable) Equation (3) for three processor grids.

$\|P\|$		5×5	10×10	15×15
50	n_{m}	17	65	140
	$recursions$	21	70	180
	$time/s$	0.07	1.80	14.05
100	n_{m}	19	75	160
	$recursions$	23	116	237
	$time/s$	0.46	14.33	69.67
150	n_{m}	20	80	–
	$recursions$	29	176	–
	$time/s$	1.57	43.11	–

Table 2. Results for testing the k-bindability equation (satisfiable) Equation (5).

$\|P\|$		$k=1$	$k=2$	$k=3$	$k=4$
30	$recursions$	434	3235	11931	35222
	$time/s$	0.12	0.94	3.62	10.69
40	$recursions$	587	4972	19136	49858
	$time/s$	0.23	1.78	7.06	20.26
50	$recursions$	633	4893	17818	53272
	$time/s$	0.37	2.74	10.02	29.90

k-Bindability Table 2 shows the average results (100 samples each) obtained from solving Equation (5) with $k = 4, \ldots, 1$ using the QBF-solver QSOLVE. We have chosen a 4×4 processor grid, different numbers $|P|$ of processes, and fixed parameters $n_{\mathrm{m}} = 13$ and $pb = 0.5$.

Since we impose only k resources to fail simultaneously, it is not necessary to test $2^{|R|}$ assignments to the variables t. QSOLVE automatically avoids searching the branches below variable assignments of (t) with more than k variables t_i set to false: After the assignment of any k variables t_i to false $f^{(k)}$ reduces to a formula f'. The universally quantified variables t_i which have not been assigned a value so far are monotonic in f'. As a result, QSOLVE searches only those parts of the search space which correspond to tuples (t) having $\leq k$ variables t_i set to false.

5. Conclusions

We considered the integration of SAT-techniques into the task of system synthesis by reducing the binding problem to quantified boolean formulas and applying the QBF solver QSOLVE. Due to short computation times, these techniques are applicable and helpful during system synthesis and easy to integrate in such design methodologies.

References

[1] 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, Jan. 1998.

[2] R. Feldmann, B. Monien, and S. Schamberger. A Distributed Algorithm to Evaluate Quantified Boolean Formulas. In *Proc. of the 17th Nat. Conf. on Artificial Intelligence (AAAI-00)*, pages 285–290, 2000.

[3] C. Scholl and B. Becker. Checking Equivalence for Partial Implementations. In *Proc. of 38th Design Automation Conference*, pages 238–243, Las Vegas, USA, 2001.