Exploiting Isomorphism for Speeding-Up Instance-Binding in an Integrated Scheduling, Allocation and Assignment Approach to Architectural Synthesis

Birger Landwehr, Peter Marwedel*, Ingolf Markhof*, Rainer Dömer***

*Dept. of Computer Science XII,
University of Dortmund, D-44221 Dortmund, Germany
email: {landwehr,marwedel,markhof}@ls12.informatik.uni-dortmund.de*

*** Dept. of Information and Computer Science,
University of California, Irvine, CA 92697-3425, USA
email: doemer@ics.uci.edu*

Abstract

Register-Transfer (RT-) level netlists are said to be **isomorphic** if they can be made identical by relabeling RT-components. RT-netlists can be generated by architectural synthesis. In order to consider just the essential design decisions, architectural synthesis should consider only a single representative of sets of isomorphic netlists. In this paper, we are using netlist isomorphism for the very first time in architectural synthesis. Furthermore, we describe how an integer-programming (IP-) based synthesis technique can be extended to take advantage of netlist isomorphism.

Keywords

IP-based architectural synthesis, netlist isomorphism, instance binding, normal form

1 INTRODUCTION

Early approaches to architectural synthesis used simplified cost functions for guiding the search for efficient RT-architectures. In particular, the effect of interconnections between RT-level components has frequently been neglected. The effect of this can be quite dramatic (McFarland 1987).

If interconnections have to be taken into account, RT-components must be uniquely labeled in order to identify the end points of interconnections*. Labels are elements of discrete sets, e.g. integers. Now, even if we restrict ourselves to integers, RT-structures can be labeled in a number of ways. Fig. 1 shows two RT-structures with labeled RT-components.

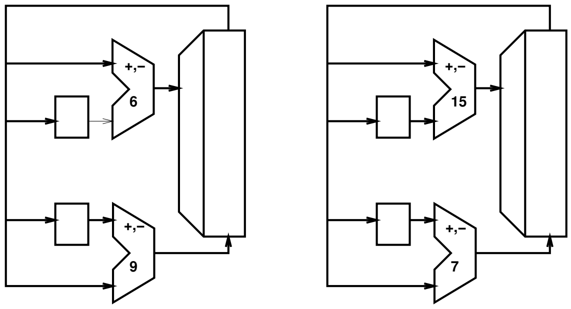

Figure 1 Isomorphic RT-Structures

Assuming that components with equal functionality are actually instances of the same component library element, these structure are obviously very "similar". In fact, we may define a function on the left structure such that its application replaces component labels by the corresponding labels in the right structure. Structures differing only by their labels are called *isomorphic*. Isomorphism has been used for graphs, finite state machines (Kohavi 1987) etc. It allows us to define the following equivalence relation:

Def. : Let n_1 and n_2 be two netlists. n_1 and n_2 are said to be **renaming-equivalent** (denoted as $n_1 \sim n_2$) if and only if there exists a bijection f on n_1 such that $f(n_1) = n_2$.

In this definition, f is supposed to replace only the component labels in the netlist that is passed as an argument. Relation $\sim$ is an *equivalence relation* and hence defines *equivalence classes*. For architectural synthesis, component labeling becomes important, if *instance binding* is considered. Instance binding is the process of binding operations of a given behavioural description to hardware components. These components are usually *instances* of library elements. Instance-binding is required if any of the following aspects are taken into account:

1. *Interconnect costs:* Architectural synthesis systems just generating bindings between operations and component types cannot model the cost of connecting component instances. Hence, interconnection costs can only be taken into account if instance binding is performed.
2. *Predefined instance binding:* Manually predefined bindings have been shown to have a positive effect on the resulting design quality (Marwedel and Schenk 1989, Arnstein and Thomas 1994). Such bindings can also be generated in interactive synthesis environments (Jerraya *et al.* 1993).

*For the sake of simplicity, we avoid the discussion about labeling component ports in this paper.

Due to the above reasons and due to the recent advances in architectural synthesis, we believe that instance binding models will be studied in more detail in the future. We will show how execution times can be shortened for these.

In particular, we will study instance binding based on an integer programming (IP) model. IP-based models exhibit a number of interesting features, including (a) the existence of a formal basis for such models, (b) the ability of integrating the three main subtasks of behavioural synthesis and a number of extensions, and (c) the fact that the model of architectural synthesis is - to a certain extent - decoupled from the algorithm implementing it. IP-based synthesis has in many cases been considered to be computationally expensive. However, we have found that IP-based synthesis is a very good building block generating extremely efficient solutions in acceptable computation time. NP-completeness of integer programming is not a problem in our synthesis system OSCAR (see (Landwehr *et al.* 1994)) based on integer programming. Runtimes remain in an acceptable range due to 1.) using all speed-up techniques of the IP model that are possible, 2.) considering only specification segments of a certain size and using the result for one segment as a starting point for the next segment, and 3.) allowing the user to switch to a relaxed linear programming model.

Due to items 2 and 3, optimality can be lost. However, the advantages of an easy integration of advanced features such as *built-in chaining* (Marwedel *et al.* 1996) are kept, regardless of which optimization mode is used.

One key speed-up technique is the contribution described in this paper. In order to illustrate how we have been able to reduce running times, consider the following typical approach to instance binding based on decision variables $x_{i,j,k}$. The variables are defined as follows:

$$x_{i,j,k} = \begin{cases} 1, & \text{if op. } j \text{ is started on component instance } k \text{ at c-step } i \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

A problem which is common to several integrated scheduling and assignment approaches (Landwehr *et al.* 1994, Hafer and Parker 1983) is the fact that the number of instances of a certain component type is usually unknown before scheduling. Hence, for each component type m, a certain number of "potentially required" instance indices $\{k_{m,1}, k_{m,2}, ...k_{m,u_m}\}$ must be used in the IP-model. In this context, u_m denotes an upper bound on the number of instances of m. Upper bounds u_m may be known for a variety of reasons. They may have been defined by the user, computed from the DFG, or computed from the costs of previously generated faster solutions.

Now suppose that operation $j = 5$ represents an addition. For the sake of simplicity let us assume that an ASAP/ALAP analysis has revealed that $i = 2$ will be the only feasible control step for $j = 5$. Also, let us assume that an upper bound of 3 has been computed for the number of adders and that instance names 6,7 and 9 have been reserved for these adders. Then, either $x_{2,5,6}$, $x_{2,5,7}$ or $x_{2,5,9}$ have to be 1 for the final design (see fig. 2).

Let us now assume that our upper bound for the number of adders was

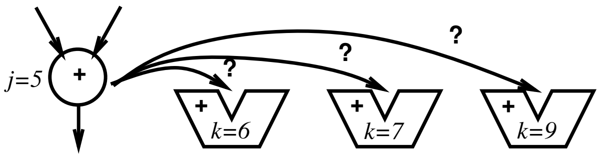

Figure 2 Possible bindings for $j = 5$

not tight, and that only two adders are actually required. Then, common instance-binding models will allow three types of solutions: a) solutions in which adders 6 and 7 are present, b) solutions in which adders 6 and 9 are present, and c) solutions in which adders 7 and 9 are present.

Obviously, it would be sufficient to consider only one type of solutions, because all the others will be equivalent in the sense defined above.

Note that, in this case, we can reduce the complexity by a factor of three by exploiting isomorphism. Larger savings can be obtained if there are more operations or component types. For example, a large number of potentially required instance names has to be reserved if there are adders with different speeds. With these savings, we can avoid one reason due to which the solution space a first sight seems to explode exponentially.

The reduction of complexity is not only possible for the model proposed by Gebotys. Actually, according to our knowledge, none of the publications on instance binding mentions techniques for taking advantage of isomorphism.

The remainder of this paper is organized as follows: Section 2 describes related work. Section 3 introduces the notation for our mathematical synthesis model. In section 4, we explain how isomorphism can be exploited in our model. Section 5 lists some practical results. The paper ends with a conclusion.

2 RELATED WORK

One of the early approaches to architectural synthesis is the IP-model of Hafer (Hafer and Parker 1983). The model does not use any kind of *normal form* for labeling components and hence implicitly analyses multiple solutions which are isomorphic.

Component labeling is also used in an approach based on simulated annealing (Devadas and Newton 1989). In that approach, operations are rebound to various control steps and RT-components. Again, the model does not use any kind of *normal form* for labeling components and isomorphic solutions are analysed.

Just like in the case of Hafer's IP model, Gebotys' approach to interconnect minimization (Gebotys and Elmasry 1991) using integer programming does not take advantage of isomorphism.

Also, no information about normal forms for component labeling is available for an interconnect minimization method based on a 3D-representation of the binding model (Stok 1990).

Note that these systems have placed the very much needed emphasis on interconnect minimization. The proposed technique aims at providing speed-ups for exactly these types of systems.

The current paper demonstrates the key concept by using integer programming as an example. However, the same concept can also be used with other approaches for searching solution spaces.

3 SYNTHESIS MODEL

3.1 General Definitions of Terms

Synthesis defines a mapping from behavioural descriptions to structural descriptions. This has frequently been described by arrows in the so-called Y-chart (Gajski and Kuhn 1983) describing the different domains in electronic design (see fig. 3).

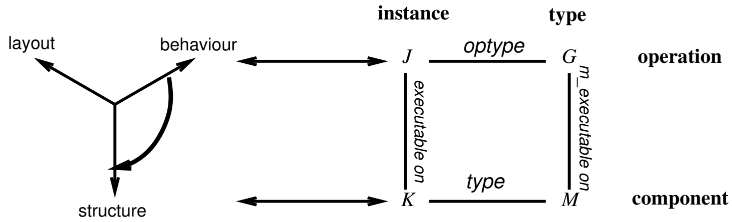

Figure 3 Naming conventions

We assume that the *behaviour* of the system under design is defined by a *dataflow-graph (DFG)*. The nodes of this graph denote operations such as additions and multiplications. More precisely, these nodes contain instances of operation types, such as "+" or "*". Let the nodes of the dataflow graph be uniquely labeled with integers from the corresponding index set $J = \{1..j_{max}\}$. We will use j as a variable to denote such integers. Furthermore, let used operation types be characterized by an index set G. Let function *optype* denote the operation type of DFG-nodes (see fig. 3).

Furthermore, we assume that the *structure* is described by a netlist containing component instances $k \in K = \{1..k_{max}\}$. Each instance is inherited from a corresponding library component type. Let variables $m \in M$ denote library component types. Function *type* denotes the type of a certain instance:

$$type : K \rightarrow M$$

The functionality of component type m is described by relation *m_executable on*:

$\forall m \in M, g \in G : g \; m\_executable \; on \; m \iff$ component type m is able to perform operation g (this information is available from the library).

From this relation, we derive the corresponding relation *executable on* among instances:

Def.: $j \in J \; executable \; on \; k \in K \iff optype(j) \; m\_executable \; on \; type(k)$.

Note that *executable on* and *m_executable on* are relations, not functions. There may be several matching component types for each operation type and vice versa. This means: our model supports a general library, including multi-functional units, mixed-speed operators, pipelining etc.

Most synthesis tools do not only generate structure. They also generate a binding between operations and *control steps* in which they are started.

The synthesis task can now be modelled as the problem of binding each operation j to a starting control step i and an executing resource k. In the following, we will show how the complexity of this problem can be reduced by exploiting isomorphism. For the sake of simplicity, we will use a normal form for labeling component instances.

4 EXPLOITING ISOMORPHISM

As a first step, we require that the set of instance indices $\{k_{m,1}, k_{m,2}, ...k_{m,u_m}\}$ forms a contigous range of integers. This means that, without loss of optimality, we restrict ourselves to functions *type* which are step functions. Moreover, we restrict ourselves to *increasing* step functions: We define ℓ_m and r_m for each $m \in M$ as

$$\ell_1 = 1 \tag{2}$$

$$r_1 = u_1 \tag{3}$$

$$\forall m > 1 : \ell_m = (r_{m-1} + 1) \tag{4}$$

$$\forall m > 1 : r_m = (\ell_m + u_m) \tag{5}$$

Then, *type* can be defined as $type\ (k) = m \iff \ell_m \leq k \leq r_m$

The next step for exploiting isomorphism is to restrict ourselves, without loss of optimality, to solutions in which integer label $\ell_m + n$ is used only if there are n or more instances of type m. This means "**lower indices are used first**". This can be expressed easily, if the presence or non-presence of a component with a certain index is explicitly modelled. For example, in our synthesis system OSCAR (see (Landwehr *et al.* 1994)), presence of instance k is modelled by a variable b_k:

$$b_k = \begin{cases} 1, & \text{if instance } k \text{ is present in a solution} \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

A straightforward approach for using "lower indices first" could consist in increasing the costs of components by a very small amount. One could define, for example, the cost of the nth component of type m as:

$cost\ (\text{instance } \ell_m + n) = cost(\text{type } m) + n * \epsilon$

Where: $\forall m, n : n * \epsilon < cost(\text{type } m)$

In contrast, we propose another method for "using lower indices first". In (Marwedel *et al.* 1995) we show that the run-time of our approach is significantly smaller than the straightforward approach. In our approach, we use additional constraints.

With additional constraints, it is quite easy to use "lower indices first". We just have to add the following constraints*:

$$\forall m \ \forall k \in [\ell_m..(r_m - 1)] \ : \ b_k \geq b_{k+1} \tag{7}$$

Limitations: The current approach to using a kind of *normal form* for labeling components assures that, for each set of isomorphic netlists, only a single representative is considered. The concept of renaming-equivalent netlists can be extended into a more general concept of equivalence. For example, our approach does not catch effects of "equivalent" wiring.

5 RESULTS

Using our OSCAR system as an example, we have analysed the actual speed-up. In order to speed up synthesis, we used a cost function considering only the cost of functional units. Constraints included: precedence constraints, functional unit constaints, assignment constraints and the renaming constraints (see (Landwehr *et al.* 1994) for details).

For the elliptical wave filter benchmark and another example for computing determinants, the speed-ups were measured on a Sun SPARCstation-20 running at 60 MHz.

We considered three different IP-solvers:

- **lp_solve, version 2.0.1**:
  Program lp_solve from the University of Eindhoven (Berkelaar 1992) is able to solve mixed integer/linear programs. We have modified the original version 2.0 such that variables b_k are considered first.
- Beta-release of **osl_solve, release 2**:
  The next solver which we considered, is based on the commercial OPTIMIZATION SUBROUTINE LIBRARY (osl) *. For this solver, we did not specify any sequence for considering variables.
- **lp_solve, version 1.0**:
  This is an earlier version of lp_solve. This version has not been modified to consider certain variables first. Since the original version 2.0 of lp_solve does not perform any better than version 1.0 if our examples are used as input, the slower execution speed of version 1.0 is essentially caused by the fact that it is does not necessarily consider variables b_k first.

Figures 4 and 5 represent the speed-up obtained for our two examples and

---

*The first p b-variables for component m can be set to 1 and the number of additional constraints can be reduced if p is the known lower bound (Ohm *et al.* 1995) on the number instances of type m (this was not exploited in the following).

*Copyright: IBM

the case of using constraints (7). The speed-up dimension had to be partitioned in order to provide a reasonable visualization of the high speed-up values.

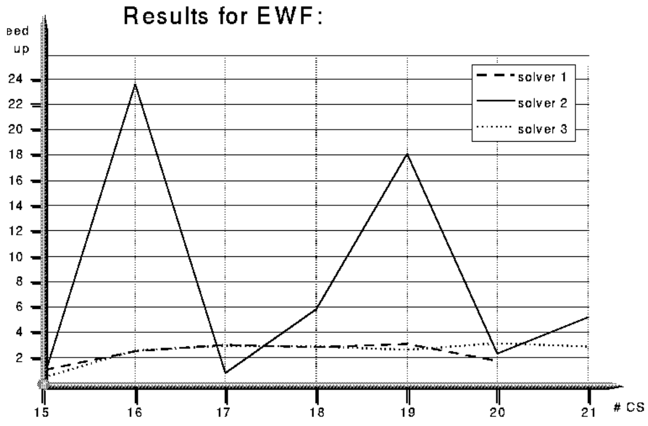

Figure 4 Summary of speed-ups for elliptical wave filter

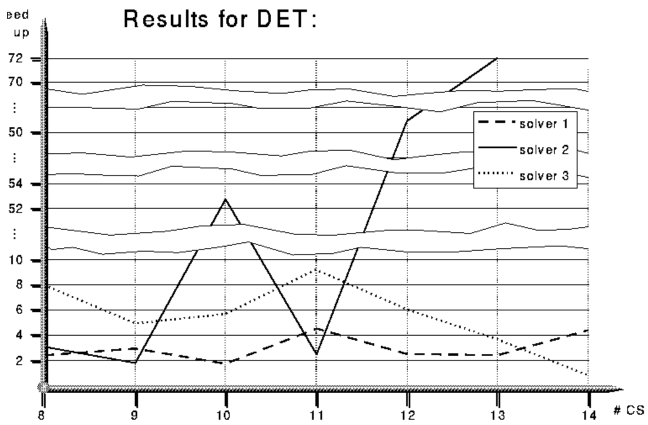

Figure 5 Summary of speed-ups for computing 3 by 3 determinantes

Note that the results have been obtained with relatively small libraries and that the speed-up is expected to increase with the size of the library.

6 CONCLUSION

In this paper, we have presented a technique for exploiting netlist isomorphism in architectural synthesis. With this technique, only one representative for each class of equivalent solutions is generated. The technique presented can be combined with a variety of synthesis models in order to reduce the run-time of architectural synthesis. Hence, the range of applications of IP-based synthesis is extended despite the fact that synthesis will still be NP-hard. Experimental results have shown that, for the examples we considered, additional constraints result in a larger runtime reduction than cost function modifications. In the OSCAR system, the technique made synthesis of larger examples feasible.

We believe that the concept of netlist equivalence reaches well beyond the current approach. In particular, it is rather straightforward to apply the approach proposed in this paper to non-IP based synthesis systems. Also, the current technique has applications in mapping computations to processors in a multi-processor system. By restricting design systems to consider only the *essential* decisions, (decisions which may actually affect the result), a lot of computation time can be saved. This might be the right way to go in order to improve phase-coupling in different design systems.

Finally, we would like to mention one recommendation for modeling. Our results clearly indicate that design constraints (such as constraints for labeling components) should be modeled as constraints and not as additional terms in the cost function. The latter of the two approaches results in increased run-times. It looks like this seemingly obvious observation is frequently ignored.

REFERENCES

Arnstein, L. F. and Thomas, D. (1994) The Attributed Behavior Abstraction and Synthesis Tools, *31th Design Automation Conference, 557-561*

Berkelaar, M.R.C.M. (1992) UNIX^{TM} Manual Page of LP_SOLVE, *Eindhoven University of Technology, Design Automation Section*

Devadas, S. and Newton, R. A. (1989) Algorithms for Allocation in Data-Path Synthesis, *IEEE Trans. on CAD, vol. 8, 768-781*

Gajski, D. D. and Kuhn, R. H. (1983) New VLSI Tools, *IEEE Computer, 11-14*

Gebotys, C. H. and Elmasry, M. I. (1991) Simultaneous Scheduling and Allocation for Cost Constrained Optimal Architectural Synthesis, *28th Design Automation Conference, 2-7*

Hafer L. and Parker A. C. (1983) A Formal Method for the Specification, Analysis and Design of Register-Transfer Level Digital Logic, *IEEE Trans. on Computer-Aided Design, Vol. 2, 4-18*

Jerraya, A. A. and Park, I. and O'Brien, K. (1993) AMICAL: an Interactive High Level Synthesis Environment, *Proceedings EDAC, 58-62*

Kohavi, Z. (1987) Switching and Finite Automata Theory, *Tata McGraw-Hill Publishing Company, New Delhi, 9th reprint*

Landwehr, B. and Marwedel, P. and Dömer, R. (1994), OSCAR: Optimum Simultaneous Scheduling, Allocation and Resource Binding Based on Integer Programming, *Euro-DAC'94*

Marwedel, P. and Schenk, W. (1989) Improving the Performance of High-Level Synthesis, *Microprogramming and Microprocessing, Vol.27, 381-388*

Marwedel, P and Bashford S. and Dömer R. and Landwehr B. and Markhof I. (1995) A Technique for Avoiding Isomorphic Netlists in Architectural Synthesis, *Report #95-28, Dept. of Information and Computer Science, University of California at Irvine*

Marwedel, P. Landwehr, B. and Dömer, R. (1996) Built-in chaining: Introducing Complex Components into Architectural Synthesis, *Report 611, Computer Science Dpt., University of Dortmund*

McFarland, M. C. (1987) Reevaluating the Design Space for Register Transfer Level Synthesis, *IEEE Int. Conf.on Computer-Aided Design(ICCAD), 262-265*

Ohm, S. Y. and Kurdahi, F. J. and Dutt, N. and Xu, M. (1995) A Comprehensive Estimation Technique for High-Level Synthesis *Int. Symp. on System Synthesis (ISSS)*

Stok, L. (1990) A Generalized Interconnect Model For Data Path Synthesis, *Proc. CompEuro 90, Tel Aviv, 461-465*

7 BIOGRAPHY

Birger Landwehr studied computer science and theoretical medicine at the university of Dortmund (Germany) and the Ruhruniversität Bochum, respectively. In 1991 he received his diploma degree in computer science from the university of Dortmund. Since 1991 he is employed as assistant at the research group "Methodology for computer-aided design of integrated circuits". His current research activities are concerned with high-level synthesis and intelligent component library mapping.

Peter Marwedel (M'79) received his Ph.D. in Physics from the University of Kiel (Germany) in 1974. Since 1989 he is a professor at the Computer Science Department of the University of Dortmund (Germany). His current research areas include hardware/software codesign, high-level test generation, high-level synthesis and code generation for embedded processors. Dr. Marwedel is a member of the IEEE Computer society, the ACM, and the Gesellschaft für Informatik (GI).

Ingolf Markhof studied computer science at the university of Dortmund in Germany. In 1990 he received his diploma degree in computer science. Since 1990 he is employed as assistant at the research group "Methodology for computer-aided design of integrated circuits". His current research activities are concerned with performance driven design for FPGA's.

Rainer Dömer studied Information and Computer Science at the University of Dortmund, Germany. He specialized in Computer Architecture and received his diploma in 1995. Since January 1996 he is working in a cooperative project between University of Dortmund and University of California, Irvine, in the field of Computer Systems Design.