Pattern Search in Hierarchical High-Level Designs

Zvi Terem, Gila Kamhi, Moshe Y. Vardi*, Amitai Irron

Logic Validation Technology, Intel Corp., Haifa, Israel
Rice Universiy*, Houston,USA

Abstract

The main focus of this paper is on using algorithms for design pattern matching to address the challenges of designs at RT and higher abstraction levels. The crux of our approach is modeling designs and patterns as graphs, which lets us express design pattern matching as subgraph isomorphism. We apply a constraint-satisfaction approach and address the problem of both exact and generalized matching. Our experimental results confirm the applicability of our approach on industrial test cases.

1 Introduction

With the introduction and acceptance of higher-level design entries, we witness several scenarios where different modeling levels need to co-exist. For example, with a detailed implementation-level model as a starting point, a designer might create a more abstract model in order to increase simulation speed. Bottom-up abstractions can simplify the logic and ease the comprehension of the functionality. On the other hand, top-down transformationss transform specification-level models to more concrete, implementable models.

In this paper, we introduce a generic search mechanism, called QUEST, of patterns at any level of design abstraction (e.g., high-level, RTL, gate-level, etc.). Based on our experience, such a generic mechanism can address several challenges of real-life micro-processor design and verification. We describe some concrete application examples. Formal verification engineers need to raise their understanding of the design in order to develop proof strategies. Furthermore, interaction with the design may be needed to reduce the model size in order to meet the capacity limitations of the downstream model checking tools. Exploration, reduction and abstraction of the design can be achieved by search of patterns of interest and replacement of the areas of interest by reduced or abstracted logic. Another example of potential use is detection of similarities and differences between two versions of a design in order to adapt existing proofs on one version to a modified version. In addition to verification problems, several design problems such as early exploration and customized synthesis can be addressed by a generic interactive search mechanism.

Design transformations and search in QUEST are performed via rules, describing how a subdesign of a particular pattern (i.e., *from-side* of the rule) should be replaced by a subdesign of another pattern (i.e., *to-side* of the rule). In case no rule for replacement is provided, QUEST reports all the patterns that match the *from-side* of the rule. The concept of transformations via rules is not new. It arose first in the context of technology mapping [K87], and have found many applications, e.g., in power optimization [RKW96]. Unlike these earlier works, QUEST facilitates interactive search and application of transformations. Therefore it addresses the problem of design exploration rather than synthesis. Moreover, we allow here general design patterns, rather than restricted ones, such as bounded degree DAGs. Using such transformations requires the ability to search for general patterns in hierarchical high-level designs. The main focus of this paper is on using algorithms for design pattern matching to address the challenges of designs at RT and higher abstraction levels. Furthermore, we demonstrate how early design exploration and synthesis problems (e.g., resource sharing) can be viewed as a special case of the general problem of design pattern matching. The crux of our approach is to model designs and patterns as graphs, which lets us express design pattern matching as subgraph-isomorphism.

Following [LV00], we use the constraint-satisfaction approach to the subgraph-isomorphism problem. The input to a constraint-satisfaction problem consists of a set of variables, a set of possible values for the variables, and a set of constraints between the variables; the question is to determine whether there are assignments of values to the variables that satisfy the given constraints, and to find such assignments. In its full generality, constraint-satisfaction is an NP-complete problem, but a wealth of heuristics enables efficient solutions in many practical cases.

We demonstrate that using the constraint-satisfaction approach to solve the subgraph-isomorphism problem enables efficient detection of all the occurrences of an arbitrary pattern in a hierarchical design. A major advantage of the constraint-satisfaction approach is its flexibility. For example, we can relax the exact matching requirement in a variety of ways. One relaxation that we found quite useful is not requiring exact match of node labels but allowing for more flexible matching predicates.

This paper is organized as follows. Section 2 summarizes related work. In Section 3, we present our technical approach on pattern search in hierarchical high-level designs and illustrate the generality of our implementation and its potential usage in solving architectural optimizations. Section 4 summarizes the results providing walkthroughs using real-life test cases. In Section 5, we present our conclusions.

2 Related Work

Graph pattern matching was first studied in a technology-independent setting by Ohrlich et al. [OEGS93], who focused on graph pattern matching as a subgraph-isomorphism problem. Their algorithm, called SubGemini, is an extension of the iterated partitioning algorithm used by the Gemini algorithm [OEGS93] for graph-isomorphism. In spite of the NP-completeness of subgraph isomorphism, a typical running time for SubGemini is approximately linear in the number of matched subgraphs. For a general discussion of graph pattern matching in computer-aided design, see [C95]. (We note that the focus in [LV00] and [C95], as well as our focus here is on syntax-based matching, because of its generality and flexibility. In certain applications it is appropriate to consider also semantic matching, which requires Boolean reasoning techniques, see [DWWC98].)

An earlier approach to subgraph isomorphism [U76]. is based on constraint-satisfaction techniques (cf, [D03]). This approach was pursued further by Larrosa and Valiente [LV00]. They evaluated several constraint-based algorithms for subgraph isomorphism, including one that is based on a look-ahead technique, and demonstrated experimentally its effectiveness across a broad range of problem instances.

From our perspective, we found the constraint-based approach better suited to our application domain. While specialized approaches, such as the one used in SubGemini, may perform better, the constraint-based approach offers greater generality and extensibility. For example, as we show later, we extended the constraint-based algorithm to handle also approximate matching,

pruning predicates, pin equivalence and the like. As we show, these extensions can all be expressed naturally in the constraint-satisfaction framework. Thus, we used Larrosa and Valiente's work as the starting point for our implementation.

3 Hierarchical Pattern Search through QUEST

In this section, we explain in detail how we reduce the hierarchical pattern search problem to subgraph isomorphism and provide a solution based on constraint-satisfaction techniques.

For the sake of clarity, we demonstrate the pattern-specific compilation and search stages of QUEST through an example. Consider the following CAD problem – "Detect all chains of two interesting boxes in a hierarchical design, at any level of abstraction". Perhaps such chains can be replaced with a more efficient single box. In this case, the design depicted in Figure 1 will be given as input to QUEST, as well as the pattern depicted in Figure 2. The expected output of QUEST will be two matches of such chains, spanning the boundary of the original module hierarchies.

3.1 Subgraph-Isomorphism Formulation of Pattern Search

Subgraph isomorphism decides if a given graph is isomorphic to a subgraph of another given graph. Although subgraph isomorphism is known to be NP-complete [GJ79] and therefore intractable, practical CAD applications [OEGS93] demonstrate that designs have sufficient structure to allow efficient solutions. A formal definition of subgraph-isomorphism is as follows:

Let $G_1 = \{V_1, E_1, L_1\}$ and $G_2 = \{V_2, E_2, L_2\}$ be two labeled graphs, where V_i is the set of vertices, E_i is the set of edges, and Li is an assignment of labels to vertices. We say that G_1 is isomorphic to a subgraph of G_2 if there is a mapping $h:V_1 \rightarrow V_2$ such that

1. *For every vertex v in V_1 we have that $L_1(v)=L_2(h(v))$, and*
2. *For every pair u,v of vertices in V_1 we have that (u, v) is in E_1 and only if (h(u), h(v)) is in E_2.*

The subgraph-isomorphism problem is to determine whether a given graph G_1 is isomorphic to a subgraph of a given graph G_2.

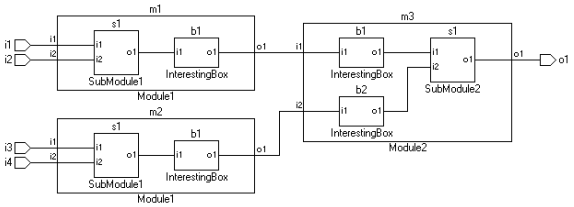

Figure 1: Simple 3-module design

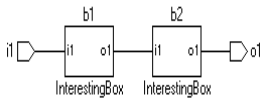

Figure 2: Simple pattern

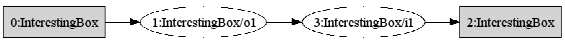

Figure 3: Graph representation of the pattern

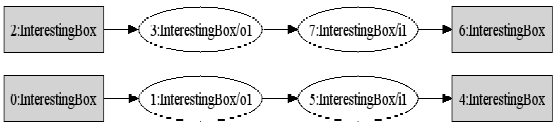

Figure 4: Reduced graph representation of the design

In QUEST, we identify the hierarchical pattern search problem with the subgraph-isomorphism problem for graphs representing the pattern and the design. The design and pattern graphs can be viewed as a directed bipartite graphs, where the design elements (which we call *instances*) are represented by squares and the interface signals (which we call *pins*) are represented by circles as shown in Figure 3.

The graph representations G_1 and G_2 for the pattern and the design respectively are generated by executing the following steps:

1. *Define the vertices representing the set of instances in both graphs when all the $v \in V_1$ in G_1 represent the top-level instances of the pattern and $v \in V_2$ in G_2 represent only the pattern instances (see discussion below for details)*
2. *Define the vertices representing the set of pins in both graphs. First include only pins connected to generated instances, then discard pins that are not on the same net (i.e. connection) as any other selected pin*
3. *Create vertices for all generated instances and pins and label them accordingly (see Section 3.3 for details)*
4. *Connect pins to their instances by edges*
5. *Connect pins to each other if they are on the same net, regardless of hierarchical boundaries*

Step 2 ensures a minimal graph with no spurious edges while Step 5 enables finding patterns that span hierarchical boundaries.

When processing the pattern, we do not drill-down beyond the top-level instances. This is based on the assumption that the instance labeling algorithm ensures that all instances with the same label represent the same function. We also record the list of instances produced in this step (which we call 'interesting' instances). When processing the design, we read the list of 'interesting' instances generated in the pattern graph. We process the entire design hierarchy down to 'interesting' instances, and produce a graph with only those instances and their net connections. Obviously, if generalized label matching predicates are used, the selection of 'interesting' instances in the design must accommodate them as well. This yields a graph with minimum size and complexity and greatly improves the search efficiency.

Another important aspect of a usable search mechanism is to correctly model pin equivalence (or pin symmetry). Many instances represent commutative operators that are not sensitive to the order of the inputs (e.g. an AND gate, or a plus operator). In such cases we model all the equivalent pins as a single graph node and connect all the individual inputs to it. This modeling approach makes sure that the search algorithm considers all possible input orderings as acceptable matches. QUEST contains a built-in description of the pin-equivalence sets for the standard gates and cells used to model designs that are input to QUEST. In general, the user can provide additional specific information on the input design elements to enhance the quality of the search.

QUEST translates the hierarchical representation of the design and the pattern, in Figure 1 and Figure 2, respectively, into a graph, with all the enhancements described above (i.e. graph reduction, generalized label matching and pin equivalence). The resulting reduced graph is illustrated in Figure 4. The reduced graph representation of the design considers only instances in the pattern (in our case – InterestingBox). In contrast, consider the full-graph representation. Note that in this case the reduction almost obviates the need for the search. In other cases, where the pattern covers more instance types, the reduction may be less dramatic.

Based on our experience, we assert that it is more efficient to re-create the design graph before each pattern search (based on the pattern instances) than to create the full graph once and use it with many patterns. In the example above, a comparison of the size of the full-design graph versus the size of the reduced design graph justifies our claim.

3.2 Constraint-Satisfaction Formulation of Subgraph-Isomorphism

An isomorphism of a graph $G_1 = (V_1, E_1)$ to a subgraph of a

graph $G_2 = (V_2, E_2)$ is equivalent to the following constraint-satisfaction problem. A variable *I* is associated with each vertex $v_i \in V_1$, and all variables take values on the domain V_2. Finding a subgraph isomorphism is then equivalent to finding a complete assignment satisfying the following *structure constraint* on the possible values assigned to i,j:

$R_{i,j} = \{(v_a, v_b) \in V_2 \times V_2 \mid v_a \neq v_b \wedge \text{edge}(G_1, i, j) \Rightarrow \text{edge}(G_2, v_a, v_b)\}$ for i, j in V_1 with $i \neq j$.

Our subgraph-isomorphism implementation is an adaptation of the constraint-propagation solution of Valiente [LV00] when the above constraint-satisfaction formulation of subgraph-isomorphism has been enhanced with the neighborhood constraint, which expresses the fact that a vertex (variable) $i \in V_1$ can only be mapped to another vertex (value) $v_a \in V_2$ if all vertices in the neighborhood of i can be mapped to other vertices in the neighborhood of v_a. The essence of the algorithm consists of two steps:

I. *Given pattern graph $G_1 = (V_1, E_1, L_1)$ and design graph $G_2 = (V_2, E_2, L_2)$, for all $v_i \in V_1$ generate candidate list D_i of matching vertices in G_2,,*
II. *SolveCSP(V_1, D) (see* [LV00] *for details)*

The CSP formulation of subgraph isomorphism provides a very natural solution to generalized pattern detection. In the first stage of the algorithm, the candidate list generation is constrained by the generalized pattern. All the candidates not satisfying the generalized pattern predicate are pruned out. The second stage of the algorithm (i.e., *SolveCSP*) is executed as usual without any changes. The easy adaptation of the solution to generalized patterns is one of the main advantages of the QUEST approach in comparison to previous work [OEGS93].

3.3 Pattern-based Design Graph Labeling

The efficiency of the search algorithm is directly related to the ability to recognize that vertices representing instances in both design and pattern graphs match. Thus, the labeling of the instances is important to uniquely identify the vertices in the graph. In some applications (like searches in a transistor-level net-list) the problem is trivial, since the name of the elements (low-level electronic devices) uniquely identifies them. The instance name might not be enough to identify an instance in a higher-level description, and it can even lead to wrong results. For example, there can be two modules labeled “filter” that perform different functions.

When the simple names of instances are not sufficient to uniquely identify them, we create a label by applying a hashing formula to the instance name, the names of the interface connections, and the labels of all sub-elements, recursively. This method is expensive, and care should be taken to perform the minimum calculation that gives satisfactory differentiation. Also, labeling of the entire design should be done in a single bottom-up sequence to prevent recalculation of labels of lower-level instances. However, in most practical cases, instance labels themselves are sufficient to identify candidate matches between pattern and design vertices, advocating the general use of a more straight-forward and efficient labeling mechanism. Since there is no automated way to recognize the need for calculating recursive labels (which we call *signatures*), it is the responsibility of the user to initiate this process.

In more complex cases as generalized patterns, a candidate match can be defined by applying a Boolean predicate between the labels of the pattern and the design nodes. In principle, there is no restriction on what that predicate can be. In practice, QUEST contains a library of built-in predicates taking some parameters, or hints, from the pattern and the design when the user can select which predicates to activate. For example, one of the built-in predicates is treating pattern instance labels as regular expressions (e.g., if a pattern instance is labeled “m\d+”, it will match any design instance label starting with ‘m’ followed by any number of decimal digits (“m1”, “m432”, etc.)).

Constraints can be applied to edges to further direct the search algorithm. For example, a common resource-sharing rule replaces the logic “out = condition ? (a + b) : (c + d)” with the more area-efficient logic “out = (condition ? a : c) + (condition ? b : d)”. Clearly, this rule can be applied to any pair of arithmetic operators (e.g., multiplication, subtraction, etc.). The application of this rule through QUEST requires first the search for the pattern “out = condition ? (a *op* b) : (c *op* d)” where *op* can be any arithmetic operator like an adder, a subtractor or a multiplier, as long as both operators are the same. A search using generalized label matching finds all the places where any two arithmetic operations are preceded with a multiplexer or “if”. The edge constraint can limit the search to cases where the instances on both ends of the edge have the same label.

QUEST allows the user to filter/constrain its output (i.e., matches for the exact or generalized input patterns) by specifying constraints through predicates. For example, in the two-interesting-box pattern search, an output filter removes cases where the net connecting the boxes is fanned out to other devices (so, for example, two inverters cannot be reduced to a wire).

In the constraint-satisfaction algorithm for finding subgraph-isomorphism, the vertex labels are used only at the start of the algorithm to generate candidate design vertices for each pattern vertex. This is particularly useful in the context of the previous paragraph – instance labels are compared once in the preprocessing stage. In contrast, labeling algorithms such as the one employed by SubGemini are less amenable to such manipulations of the instance names in the preprocessing stage.

4 Experimental Results

4.1 Exact Matching

The highly interconnected complex structure of multipliers challenges state-of-the-art synthesis and verification solutions. Thus we have chosen first as our design an implementation of an integer multiplier. Table 1 below presents the results of QUEST in search for a simple gate pattern in the hierarchically synthesized version of the integer multiplier, which is scaled to different operand widths. Note the drastic reduction in graph size (2-3X) and the direct effect of graph size and number of matches on CPU time.

In order to demonstrate the usage of QUEST in the application of bottom-up transforms, we have chosen the problem of searching for a basic 1-bit ripple-carry adder pattern in a synthesized, flattened 256 bit ripple-carry adder implementation. The design at hand is an implementation of an N-bit ripple-carry adder that consists of N replications of the basic 1-bit adder pattern. QUEST successfully facilitated abstract representation of the adder when the ripple carry adder logic has been identified and abstracted out to separate logic blocks. The reduced graphs for 32-256 bit adder varied from 1053 to 8445 edges.

In contrast, consider searching for a much simpler pattern consisting of a 3-way OR connected by one pin to a 3-way XOR (represented by a graph consisting of 4 vertices). This pattern spans the boundary of the basic cells and thus occurs (width – 1) times. Note the drastic reduction in the graph size, and QUEST time in the results reported in the Table 2 below where the reduced graph size varies between**.**

4.2 Generalized Matching

In order to demonstrate the ability of QUEST to find generalized patterns, let us consider the “resource sharing” transformation explained in Section 3.3. The application of this rule through QUEST requires first the search for the pattern “out = condition ? (a *op* b) : (c *op* d)” where *op* is constrained to be any binary

arithmetic operator (e.g., adder, subtractor or multiplier) as long as both operators are the same. All the locations in the design that satisfy this pattern can then be replaced by an adder saving pattern "out = (condition ? a : c) + (condition ? b : d)".

	Full Graph		Reduced		Number	Quest
Width	Nodes	Edges	Nodes	Edges	Matches	Time
32	30,646	87,697	13,383	18,253	900	1:30:00.0
28	23,162	64,177	10,049	13,648	678	34:20.0
24	16,838	45,288	7,217	9,822	486	11:13.0
20	11,592	30,419	4,872	6,649	326	03:17.0
16	7,196	18,219	2,957	4,008	196	00:43.0
8	1,610	3,909	585	805	36	00:00.4

Table 1: Presents search results of a simple gate pattern in an integer multiplier.

	Without pin equivalence Reduced Graph			With pin equivalence Reduced Graph		
Width	Nodes	Edges	Time(s)	Nodes	Edges	Time (s)
32	128	94	0.1	161		0.1
64	256	382	0.2	321		0.2
128	512	511	0.7	641		0.8
256	1,024	766	5.2	1,281		5.0

Table 2: Presents the results of QUEST in search of a simple gate pattern in a 32, 64, 128, 256 bit adder implementation.

We have generated a scalable design to measure the performance of QUEST in interactive detection of the resource sharing opportunities in a high-level RTL design (consisting of multiplexers, ALUs, etc.) as can be seen in Figure 5.

The basic block depicted below in Figure 5 contains the *from-side* of the resource-sharing transformation. Clearly, in every block there are 3 matches (i.e., two matches with adder operator in the sub-block 'incr' and one with subtractor operator in the sub-block 'absdiff'). Therefore, the number of matches is 3 times the number of blocks. Table 3 presents the reduced graph size and CPU time spent by QUEST in the detection of all the resource sharing opportunities in the designs consisting of (8-64) replications of the basic block which consists of (72-576) architectural components.

4.3 Real-life Test Cases

We have applied QUEST on two industrial designs at hybrid abstraction levels. QUEST successfully finds 82 matches of the pattern "nand (not a) (not b)" in a hierarchical real-life design (i.e., D1) that consists of 1,482 instances. 40% of the matches span hierarchy boundaries.

Furthermore, we have given QUEST the problem of finding two adder chains making use of a version of D1 which has been flattened by one level. The partial flattening accounts for the reduction in the number of instances 1334 versus 1482. QUEST finds 16 matches in less than 1 second. In a significantly larger test case D2, QUEST searches for all multiplexers with latched outputs. In a partially synthesized version of D2 (thus larger: 3781 versus 3184 vertices), QUEST finds 36 matches for a specific instance with two inverted inputs in 0.76 seconds.

5 Conclusions

We have introduced a generic exact and generalized pattern matching mechanism, QUEST, and demonstrated its applicability to solve practical CAD problems that deal with bottom-up and top-down transformations of designs at hybrid abstraction levels. To the best of our knowledge, QUEST pioneers in the application of constraint-satisfaction techniques to the problem of search of sub-design of interest in real-life CMOS test cases. Moreover, the usage of constraint-satisfaction techniques facilitates easy extension of QUEST to deal with generalized patterns, which is difficult to apply to prior related work [OEGS93]. Another important contribution of this paper is pattern-based reduction of the design, which at times obviates the needs for the search. Moreover, the generality of QUEST architecture facilitates its suitability to various CAD frameworks.

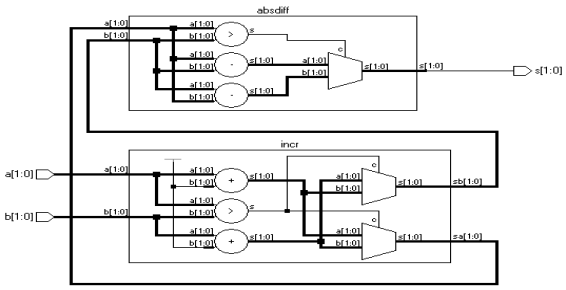

Figure 5: Basic block consisting of 9 architectural components

Blocks	Nodes	Edges	Time (s)	Blocks	Nodes	Edges	Time (s)
8	498	1,009	0.3	32	1,984	64,033	21.0
16	994	2,017	1.7	64	3,970	8,065	312.4

Table 3 : Results for resource-sharing example

6 References

[B88] Michael Boehner. "LOGEX – An Automatic Logic Extraction from Transistor to Gate Level for CMOS Technology", In Proceedings of the 25th Design Automation Conference, pages 517-522, June 1988.

[C95] T.S. Chanak. "Netlist Processing for Custom VLSI via Pattern Matching", Technical report CSL-TR-95-681, Stanford, 1995.

[CB95] S.Chandrakasan, R.W. Brodersen, "Minimizing Power Consumption in Digital CMOS Circuits", In Proceedings of the IEEE, Vol. 83, No.4, pp. 498-523, April 1995.

[D03] R. Dechter, "Constraint Processing", Morgan Kaufmann, 2003.

[DWWC98] T. Doom, J. White, A. Wojcik, G. Chisholm. "Identifying High-Level Components in Combinational Circuits.", In GLS-VLSI'98

[EZ83] C. Ebeling and O. Zajicek. "Validating VLSI Circuit Layout by Wirelist Comparison", Proc. of ICCAD-83

[GJ79] M. Garey and D.~S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-completeness", W.Freeman&Co., Francisco, 1979.

[GDWL93] Gajski, Dutt, Wu, Lin, "High-Level Synthesis", KAP

[K87] K.Keutzer, "DAGON: Technology Binding and Local Optimization by DAG Matching", Proc. ACM/IEEE DAC, 1987, pp. 341- 347.

[LV00] Javier Larrosa and Gabriel Valiente. "Graph Pattern Matching using Constraint Satisfaction", In Proc. Joint APPLIGRAPH and GETGRATS Workshop on Graph Systems Transformation, 2000.

[OEGS93] M. Ohlrich, C. Ebeling, E.Ginting and L.Sather, "SubGemini: Identifying Sub-Circuits Using a Fast Subgraph Isomorphism Algorithm", In Proceedings of ACM/IEEE Design Automation Conference, 1993.

[RKW96] B.Rohfleisch, A. Kolbl, B. Wurth. "Reducing Power Dissipation by Structural Transformations", In Proc. 33rd DAC, 1996.

[U76] J.R. Ullman, An algorithm for sub-graph isomorphism, J. ACM 23(1), 31-42, 1976.