C. Computation Cost Analysis

A comparison of the computation times between traditional harmonic balance and the proposed method for determining the sensitivity of IP3 using the adjoint approach is shown in Table I obtained using a prototype MATLAB simulator on a local workstation. As can be seen, the proposed method presents a significant speedup.

It is important to note that both approaches cannot be taken independently. In the case of harmonic balance, we must first compute the value of IP3 from a standard harmonic balance simulation in order to obtain the harmonic balance Jacobian. For both methods, the computation time for finding the sensitivity, with respect to additional circuit parameters, is coupled with the time of the original computation distortion based on adjoint moments analysis. This presents a significant speedup.

In the case of the proposed approach, we also need to obtain IP3 using the moments-based method in order to have access to the moments computation matrix. Therefore, it is more meaningful to compare the central processing unit (CPU) times for computing both the nominal value of IP3 and its sensitivity using both approaches. As a result, when the computation times shown in Table I are coupled with the time of the original moments technique for obtaining IP3 as described in [7], the result is a very efficient technique for finding both IP3 and its sensitivity with an overall speedup shown in Table II over harmonic balance. For both methods, the computation time for finding the sensitivity, with respect to additional circuit parameters, was negligible. This is a property of the adjoint method.

VIII. Conclusion

In this paper, a new method was proposed for the efficient sensitivity analysis of third-order nonlinear intermodulation distortion based on adjoint moments analysis. This new method added insight to the moments-based method for computing IP3, presented in [7], while still remaining significantly more efficient than traditional simulation approaches based on harmonic balance. The method was shown to be general and, therefore, applicable to arbitrary nonlinearities and circuit topologies. The sensitivity obtained using the proposed approach was as accurate as the harmonic balance adjoint method.

IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS 939

<table>
<thead>
<tr>
<th>Type of Circuit</th>
<th>Type of Computation</th>
<th>Harmonic Balance Time (s)</th>
<th>Proposed Method Time (s)</th>
<th>Speed-Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common</td>
<td>IP</td>
<td>0.76</td>
<td>0.25</td>
<td>3.1 times</td>
</tr>
<tr>
<td>Emitter</td>
<td>Sensitivity</td>
<td>0.81</td>
<td>0.34</td>
<td>2.4 times</td>
</tr>
<tr>
<td>Amplifier</td>
<td>Total</td>
<td>0.50</td>
<td>0.15</td>
<td>3.3 times</td>
</tr>
<tr>
<td>Differential</td>
<td>IP</td>
<td>0.67</td>
<td>0.14</td>
<td>4.7 times</td>
</tr>
<tr>
<td>Singly</td>
<td>Sensitivity</td>
<td>0.65</td>
<td>0.01</td>
<td>31.3 times</td>
</tr>
<tr>
<td>Balanced</td>
<td>Total</td>
<td>0.43</td>
<td>0.01</td>
<td>41.7 times</td>
</tr>
<tr>
<td>Mixed</td>
<td>Total</td>
<td>0.61</td>
<td>0.01</td>
<td>58.8 times</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TABLE II

Computation Cost Comparison of Finding Both IP3 and Its Adjoint Sensitivity for the Example Circuits

REFERENCES


Pattern-Mining for Behavioral Synthesis
Jason Cong, Fellow, IEEE, Hui Huang, and Wei Jiang

Abstract—Pattern-based synthesis has drawn wide interest from researchers who tried to utilize the regularity in applications for design optimizations. In this letter, we present a general pattern-based behavioral synthesis framework which can efficiently extract similar structures in programs. Our approach is very scalable in benefits of advanced pruning techniques. The similarity of structures is captured by a mismatch-tolerant metric: the graph edit distance. The graph edit distance can naturally capture different program variations such as bit-width, structure, and port variations. In addition, we further our approach to handle control-intensive applications, and this leads to more opportunities for optimization. Our algorithm uses a feature-based filtering approach for fast pruning, and a graph similarity metric called the generalized edit distance for measuring variations in control-data flow graphs. Furthermore, we apply our pattern-based synthesis system to the resource optimization problem in behavioral synthesis. Considering knowledge of discovered patterns, the resource binding step can intelligently generate the data-path to reduce interconnect costs. Experiments show that our approach can, on average, reduce the total area by about 20% with respect to the baseline.

Manuscript received December 3, 2009; revised May 18, 2010 and October 3, 2010; accepted December 9, 2010. Date of current version May 18, 2011. This work is supported in part by the SRC GRC, under Contract 2006-TJ-1400, in part by the NSF, Grant CCF-0530261, and a grant from Xilinx and Magma, under the California MICRO Program. This paper was recommended by Associate Editor S. Nowick.

The authors are with the Department of Computer Science, University of California, Los Angeles, CA 90095 USA (e-mail: cong@cs.ucla.edu; hhuang@cs.ucla.edu; wjiang@cs.ucla.edu).

Digital Object Identifier 10.1109/TCAD.2011.2106370
Pattern-based synthesis has drawn wide interest from researchers who try to extract and utilize the regularity in applications for design optimizations. The common tasks of pattern-based synthesis consist of pattern matching and pattern recognition. Pattern matching is a technique for checking the presence of a given pattern. Representative works in pattern matching include graph-parsing in cognitive studies [1], [2], symbolic equivalence checking [3], abstract syntax tree-based matching [4], [5], and graph isomorphism algorithms [6]. Pattern recognition [7] was initially studied in the machine learning domain for taking action based on the category of data, i.e., extracting patterns from the raw data. Of all the pattern recognition techniques, structural pattern recognition is a methodology which attempts to describe objects in terms of their parts and connections. Structural pattern recognition is mainly based on graph matching, where each object is represented by a labeled graph. Recently, graph matching has found many applications in data mining, biochemistry, and very large scale integrated computer-aided design.

In circuit designs, the intelligent use of regularity usually produces high quality results. Actually, this is one key reason why careful manual design can excel over the design synthesized by automated tools. In this letter, we attempt to optimize the resource usage of field-programmable gate array (FPGA) designs using pattern-based synthesis techniques.

It is not surprising that pattern recognition has been exploited in every level of the large circuit design, from layout designs to high-level synthesis [8]–[13]. Previous works on pattern matching in behavior synthesis have different limitations, such as pattern size, pattern representation (tree or string), scalability, and mismatch-tolerance. In this letter, we propose a general and efficient pattern recognition and synthesis framework which benefits from the advanced subgraph enumeration/pruning/matching techniques. Each pattern is represented by a labeled directed graph (DAG) to capture the control/data flows in the original program. In particular, the contributions of our approach include the following.

1) Use of a graph similarity metric called generalized edit distance [14] which can naturally handle various program variations such as bit-width, structure and port variations, and control flow difference. Our method can also be extended to handle edit operations with different costs for area optimization.

2) Efficient pruning techniques for pattern recognition.

3) An efficient and accurate pattern selection strategy which helps to select optimal pattern combinations from discovered patterns.

4) A pattern-based behavior synthesis flow targeting FPGA for resource reduction. With the knowledge of patterns, our approach minimizes the resource cost through pattern selection, pattern-adaptive scheduling, and binding on the basis of the target FPGA platform.

The remainder of this letter is organized as follows. Section II discusses related work; Section III extends our data flow graph (DFG)-based algorithm to a more generalized control-data flow graph (CDFG) pattern recognition algorithm. Section IV presents our overall pattern-based behavior synthesis flow; Section V reports experimental results and is followed by conclusions in Section VI.

II. RELATED WORK

Graph-matching-based pattern recognition method mentioned above has been widely applied to data mining, and a number of efficient and scalable algorithms have been developed to find frequent patterns in graphs [15]–[18]. The a-priori-based graph mining (AGM) work proposed in [18] uses a level-wise scheme to enumerate the recurring subgraphs. The pattern candidates with size $k + 1$ are constructed by joining two graphs with size $k$ which share $k − 1$ common nodes, and a frequency count of the current pattern candidate is done by subgraph isomorphism checking. The frequent subgraph algorithm proposed in [16] extended the AGM algorithm to handle connected subgraphs, and they both use a breadth-first search strategy. Another group of algorithms use depth-first search to find frequent subgraphs like graph-based substructure pattern mining (gSpan) [17] and fast frequent subgraph mining (FFSM) [15]. Both approaches calculate the canonical label of a graph to avoid redundant subgraph enumeration and graph isomorphism test; gSpan uses a canonical representation of a depth-first traversal of a graph and FFSM uses the adjacency matrix of a graph. The canonical labeling problem is NP-hard in general, and different heuristics have been proposed to incrementally construct the canonical label. All of the aforementioned work can guarantee the completeness of finding frequent subgraphs in terms of graph isomorphism.

A preliminary version of this letter was reported in [19], in which an efficient subgraph enumeration technique has been proposed to accelerate pattern recognition process and mismatches can be handled using the edit distance metric. In this letter, the original approach is extended to handle CDFGs. Compared with the previous method, it can utilize regularities across basic blocks and support sharing with multi-basic block patterns. Specifically, a hierarchical feature-based filtering scheme is introduced to effectively reduce the amount of expensive similarity evaluation computations on CDFGs. We further extend the edit distance definition in [19] to handle the matching between two sets of graphs for CDFG pattern-mining.

III. CDFG PATTERN RECOGNITION

In this section, we will first introduce four important techniques used in our CDFG pattern recognition approach. Then a generalized pattern recognition algorithm for the CDFG will be described, which can be used to discover patterns with similar CDFG structures.

A. Techniques

1) CDFG Labeling: In this letter, labeled graphs are used to describe patterns, and naturally, we can easily derive a...
At step up constructive method for the CDFG subgraph enumeration in CDFG pattern. will be used to differentiate the conditional label since it is not commutative. Similarly, different edge label same label, while edges of a same label. For example, the two input edges of a+ should have the original DFG label of the nodes it contains. When the label of a basic block is obtained, we will attach it blocks in the same group will be assigned the same label. blocks according to their internal data structures, and basic techniques discussed in Section III-A4, we can group basic treated as one supernode. With the CDFG similarity evaluation the type of the respective operation (addition, multiplication, and so on). For a control flow graph, each basic block will be treated as one supernode. With the CDFG similarity evaluation techniques discussed in Section III-A4, we can group basic blocks according to their internal data structures, and basic blocks in the same group will be assigned the same label. When the label of a basic block is obtained, we will attach it to the original DFG label of the nodes it contains.

Commutativity and other properties can be handled by edge labels. For example, the two input edges of a+ should have the same label since it is not commutative. Similarly, different edge label will be used to differentiate the conditional if and else branch in CDFG pattern.

2) CDFG Subgraph Enumeration: We propose a bottom-up constructive method for the CDFG subgraph enumeration problem. Each basic block in CDFG is treated as a supernode. At step k+1, all the subgraphs with k supernodes are generated, and they are extended by adding one neighbor in the original control flow graph. In order to reduce duplication, we define a global order of each supernode as a unique index, which will directly correspond to its extension order.

3) Two-Level Feature-Based Filter: In our approach, a signature called two-level characteristic vector (CV) is introduced for each CDFG subgraph. The DFG-level CV proposed in [19] has been extended by including CDFG feature in the previous framework to handle difference between CDFG patterns. However, the previous CV is constricted to the DFG only.

Definition 1: In a CDFG graph G = (V_G, E_G), a CDFG feature is a subgraph S = {u, l_1, ..., l_m | u, l_i ∈ V_G} ⊆ G, such that edge (u, l_i) ∈ E_G (m equals the number of outputs for supernode u).

Fig. 1 shows the features of a CDFG graph, as well as the corresponding CV. Here assume basic blocks 1 and 2 are similar, therefore, the number of the second feature in CV is 2.

The filtering techniques in [19] has the restriction that each operation must have uniform cost. In practice, each edit operation may have different costs in different applications. For example, replacing + with − is less costly than replacing + with * in hardware design. To handle these situation more precisely, weighted edit distance is introduced as following.

Definition 2: Assume S = {op_1, op_2, ..., op_n} is a sequence of edit operations which transforms a labeled graph G_1 to G_2, and the cost of each operation op_i is C_i. The weighted edit distance wd(G_1, G_2) [14] of two labeled graphs G_1 and G_2 is the minimal total cost (∑ C_i) of any possible transformation sequence.

The general weighted edit distance cannot be pruned using techniques in [19], since the edit distance is not increased by 1 for each edit operation. However, it can be extended to handle weighted edit distance assuming that the cost of each edit operation must be a positive integer. We can assign an integer weight w_i to each node n_i, then the insertion/deletion cost is C_INSERT = C_DELETE = w_i, and the replacement cost is C_REPLACE = |w_i − w_j|.

For example, in hardware design, we can use normalized area of operations as the cost, such that +/− operations are less likely be replaced by + operations with a small edit distance threshold, but they can be replaced by operators with similar sizes.

With the above restriction, we can further prove the following.

Theorem 1: Let wd(G_1, G_2) be the weighted edit distance between two DAG G_1 and G_2, CV(G_1), CV(G_2) be the CVs of G_1 and G_2, respectively. |CV(G_1) − CV(G_2)| ≤ 4 * wd(G_1, G_2). Proof: One edit operation op can at most change CV in terms of L_1 norm by 4. And since C_1 is a positive integer (i.e., C_1 ≥ 1), we have |CV(G_1) − CV(G_2)| ≤ 4 * C_1. Adding all the edit operations together, we come to the conclusion that |CV(G_1) − CV(G_2)| ≤ 4 * ∑ C_i ≤ 4 * wd(G_1, G_2).

The combination of data flow CV and control flow CV is used in our approach to capture structural properties of a given CDFG graph. Theorem 1 tells us that given an edit distance limit l_dist, the data flow CV difference between two CDFG subgraphs will not exceed 4*l_dist, namely, the maximal number of possible data flow feature misses is 4*l_dist under the edit distance constraint. However, this reveals no information for the similarity degree in the two control flow graphs.

In order to develop an upper bound for the number of possible CDFG feature misses, we propose a data structure called the feature map matrix. Each row of the feature map matrix corresponds to a DFG feature, while each column corresponds to a target CDFG feature. Each entry records whether a DFG feature appears in a target CDFG feature. As shown in Fig. 2, DFG features f_1 and f_2 appear once in CDFG feature F_1 and the corresponding entries in the feature map matrix are set to one.

Definition 3: Given edit distance limit l_dist, l_dist is defined to be the maximum number of columns covered by 4 * l_dist rows in feature map matrix.
Theorem 2: Let $l_{edit}$ be the given edit distance limit, and $CV_{CFG}(G_1)$ be the control flow CV of $G_1$. If the generalized edit distance (GED) between $G_1$ and $G_2$ does not exceed $l_{edit}$, we have $|CV_{CFG}(G_1) − CV_{CFG}(G_2)| ≤ l_{edit}$.

Proof: Assume the GED between $G_1$ and $G_2$ is less than $l_{edit}$. Theorem 1 tells us that the data flow CV distance between $G_1$ and $G_2$ is no more than $4 + l_{data}$, if a DFG feature $f_i$ is missing, all the CFG features containing $f_i$ in their inside data flow structures will be destroyed correspondingly. Therefore, their control flow CV distance $|CV_{CFG}(G_1) − CV_{CFG}(G_2)|$ cannot exceed $l_{edit}$.

Based on the analysis above, we know that subgraphs with CV difference larger than $l_{edit}$ can be filtered in advance and reduce the number of accurate similarity comparison.

4) Similarity Evaluation: Given two CDFG subgraphs $G_1$ and $G_2$ which have passed the two-level feature-based filter, similarity evaluation will first be performed between their control flow structures, in which each basic block is treated as a supernode. After that, we will look into the DFGs inside to do further comparisons.

We can observe that the data flow structure inside each supernode is not connected and consists of several separate subgraphs which we call subgraph fragments. This situation is very common. In our approach, we do not allow edge insertion operation between two separate subgraph fragments, since it will connect too originally parallel DFGs and latencies of pattern instances may differ too much. Under this constraint, we define generalized edit distance as follows.

Definition 4: Given two sets of subgraph fragments $SF_1$ and $SF_2$, they are defined as $SF_i = \{f_{i1}, f_{i2}, ..., f_{i3}\}$ and $f_{ij}$ is the $j$th subgraph fragment in set $i$. Assume the edit distance between graph $f_{i1}$ and $f_{j1}$ is $d(f_{i1}, f_{j1})$, the GED between $SF_1$ and $SF_2$ is defined as $\min_{1 \leq i, j \leq N} \sum_{k=1}^{N} d(f_{ik}, f_{jk})$, where $(p_1, p_2, ..., p_N)$ is a permutation of $(1, 2, ..., N)$.

To calculate GED between two sets of subgraph fragments, we construct a fragment-edit-distance matrix as follows. Entry $M(i, j)$ in the matrix records the edit distance between the $i$th subgraph fragment in set 1 and the $j$th fragment in set 2. When we build $M$, the condition edit distance $d(f_{i1}, f_{j1}) ≤ l_{data}$ must be satisfied, otherwise, an infinite value will be assigned to the corresponding entry. In our experiments, we find that in most cases, the number of fragments with more than ten nodes in a given subgraph is less than five; therefore, even though we need to compute edit distance between every two fragments, the cost is still acceptable. With the fragment-edit-distance matrix, our problem is to find an optimal index permutation $(p_1, p_2, ..., p_N)$ of $(1, 2, ..., N)$, so that the sum of edit distance between the $i$th fragment in the first set and the $p_i$th fragment in the second set is minimal, for $i = 1$ to $N$.

This problem can be formulated as assignment problem, and Hopfield network has been developed to solve this problem efficiently in polynomial time [20].

B. CDFG Pattern Recognition Algorithm

Our algorithm iteratively finds patterns of size $k$ starting from $k = 1$. At step $k + 1$, all the size $k$ CDFG pattern instances are extended by one supernode using the subgraph enumeration techniques discussed in Section III-A2. If subgraph $s_k$ is not a pattern instance of a certain pattern $P$ at step $k$, it is impossible for it to be a subgraph of another pattern instance larger than $k$, which means we do not need to further extend it. When a new subgraph $s_{k+1}$ is generated, it will be compared to the existing patterns by calculating the CFG level edit distance between itself and existing patterns.

First, the control flow CV of a subgraph is calculated and used as a signature to find the patterns which have similar control flow structures. After getting the list of possible pattern candidates, GEDs are calculated by the techniques discussed in Section III-A4. If $s_{k+1}$ matches a pattern $P$, it will be added to the pattern instance list of $P$, otherwise a new pattern will be generated based on $s_{k+1}$.

IV. PATTERN-BASED SYNTHESIS FLOW FOR FPGA RESOURCE REDUCTION

Our pattern recognition framework can be applied in many practical problems, such as the FPGA resource reduction problem (PBS-RR) discussed in this letter. If all pattern instances are scheduled and bounded in a uniform way, the internal data flows are free of multiplexors (except the multiplexors generated due to resource sharing among nodes inside a single pattern instance). Based on this observation, a pattern-based behavior synthesis flow is proposed in this section for FPGA resource reduction.

Specifically for the PBS-RR problem, only vertex relabelling is allowed in edit distance calculation. The reason is that vertex insertion/deletion not only increases the resource usage of a single pattern with additional multiplexors to handle variations among pattern instances but also complicates the scheduling algorithm by introducing latency variations.

Pattern selection attempts to find an appropriate set of pattern instances which minimize resource usage and latency overhead.

For the PBS-RR problem, the following metric is used for a given pattern $P$ with $N$ compatible pattern instances [19] as follows:

$$N \new{+} \max_{1 \leq i \leq N} \{area(P_i) \} \frac{\text{area}(P) + \text{area}(\text{internal}(P)) - \text{area}(P_i)}{\text{area}(P_i)}$$

With the definition of pattern gain, the problem is how to select a subset of patterns which are non-conflicting and will maximize the total gain. Here “non-conflicting” means non-overlapping, and no loop will be formed in a DFG after selecting a certain set of patterns. For example, given a CDFG graph $G$ which consists of seven basic blocks, indexed from 0 to 6, assume our pattern recognition algorithm finds three patterns $P_0, P_1$, and $P_2$ in $G$. The corresponding pattern groups are denoted by $(P_0 \{1, 2\}, P_1 \{0, 1, 3\}, 4, 5)$, and $P_2 \{3, 4\}$. That is, pattern $P_0$ has two 1-node instances, and the node index for each instance is 1 and 2, and so on. A conjunctive normal form (CNF) representation $F(p_0, ..., p_n)$ is used in our approach to describe the non-overlapping constraint among pattern group candidates. If $p_0$ is set to 1, the corresponding pattern $P_0$ will be selected. For example, the non-overlapping constraint for pattern group 0 can be represented by setting $f_0 = \neg p_0 \lor \neg p_1 \lor \neg p_0 \lor \neg p_1$.
to 1. Based on the discussion above, the final CNF constraint is \( F(p_0, p_1, p_2) = f_0 f_1 f_3 f_5 = 1 \), and our objective is to maximize total gain. With this formulation, our problem can be reduced to a binate covering problem, and the bounding technique in [21] can be used to compute the optimal solution.

After pattern selection, the scheduling and binding algorithms are fairly easily designed to solve the PBS-RR problem. Briefly, each pattern is scheduled and bound based on the resource constraints in advance to get the respective hardware implementation. Next, patterns are viewed as complex multicycle operations, and any state-of-the-art behavior synthesis algorithm can be easily adapted for PBS-RR problem.

V. EXPERIMENTAL RESULTS

Our pattern-based synthesis flow has been implemented in the xPilot behavior synthesis system [22]. xPilot takes behavioral languages like C as input and parses them into control DFGs. The control data flows graphs are viewed as collections of DFGs for pattern recognition. The graph matching toolkit [23] is used for graph edit distance calculation. The synthesis engine will then perform the pattern-based synthesis flow to reduce the resource usage with certain design constraints. The synthesis results are dumped into RT-level VHDL and accepted by the downstream register transfer level synthesis tools. Our experiments use the Xilinx Virtex-4 FPGA and ISE 9.1 tool [24].

A. Resource Reduction with CDFG Pattern Recognition

To further illustrate the efficiency of our CDFG pattern recognition algorithm, similar experimental work has been applied to six real-life test cases containing common control flow structures in the program. Our test cases include IDCT, SYNFLT, BH, BLKSORT, HEAP, and LEXTREE. Similarly, we also test the effectiveness of the proposed control-flow-involved pruning techniques on these six benchmarks. On average a very small number of GED computations is needed as observed in Table I. Line indicates the size of each test case, Pattern and Inst represent the number of patterns and total pattern instances, respectively, in Table I, where Avg.Calc is the average number of GED computations needed and MAX is the maximal size of patterns in terms of DFG nodes.

The pattern-based FPGA resource reduction results with our CDFG pattern recognition algorithm are applied in Table II. Our letter has been compared to a traditional behavioral synthesis flow without pattern optimization techniques involved and the one with DFG pattern recognition. In Table II, the second, third, and fifth columns show the synthesis results for our pattern-based FPGA resource reduction results with our CDFG pattern recognition algorithm applied in Table II. Table III shows that our pattern recognition can effectively reduce the resource usage with certain design constraints. The performance improvement is especially substantial in BLKSORT in which pattern instances are distributed among different basic blocks, while the DFG-based approach cannot efficiently deal with sharing at the basic block level. The latency overhead here is about 9% on average with 3.5% clock period increase.

B. Scalability of CDFG Pattern Recognition

With the proposed CDFG pruning techniques, our pattern recognition algorithm scales well as code size increases, which has been tested on all C programs in SPEC2006 [25]. Overall, our CDFG pattern synthesis flow has a 24% resource reduction on average compared with the traditional one, and outperforms the one with DFG pattern-based technique, respectively. Columns 7–11 list the amount of resource reduction on average compared with the traditional one, and outperforms the one with data flow pattern only based approach cannot efficiently deal with sharing at the basic block level. The latency overhead here is about 9% on average with 3.5% clock period increase.
columns 2–6 represent lines of C code, patterns found, pattern instances, size of largest patterns, and runtime, respectively. For example, test bench 445.gobmk has about 200,000 lines of C code, and the biggest pattern contains 25 operations, yet our algorithm can discover all patterns in about 20 min. In practice, with further design constraints (like size of patterns, input and output limitations, and others) and user-defined pruning criteria, we believe that runtime will not be a serious issue.

VI. Conclusion

In this letter, we presented a general pattern-based behavior synthesis framework which can efficiently extract patterns from behavior specifications. Further, the pattern recognition framework was applied to solve the resource optimization problem on FPGA platforms. Experiments showed the efficacy of both the pattern recognition algorithm and the resource reduction algorithm.

REFERENCES


