TABLE II
COMPUTATION COST COMPARISON OF FINDING BOTH IP3 AND ITS ADJOINT SENSITIVITY FOR THE EXAMPLE CIRCUITS

Type of Circuit	Type of Computation	Harmonic Balance Time (s)	Proposed Method Time (s)	Speed-Up
Common Emitter Amplifier	IP3	8.78	1.22	7.2 times
	Sensitivity	0.81	0.31	2.6 times
	Total	9.59	1.53	6.3 times
Differential Amplifier	IP3	44.67	3.45	13.0 times
	Sensitivity	6.03	0.52	11.7 times
	Total	50.7	3.97	12.7 times
Singly Balanced Mixer	IP3	118.43	4.01	29.5 times
	Sensitivity	21.63	2.59	8.4 times
	Total	140.06	6.60	21.2 times
Doubly Balanced Mixer	IP3	158.91	5.08	31.3 times
	Sensitivity	41.66	3.34	12.4 times
	Total	200.57	8.42	23.8 times

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. 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 combine 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.

REFERENCES

[1] B. Razavi, *RF Microelectronics*. Englewood Cliffs, NJ: Prentice-Hall, 1998.

[2] J. Rogers and C. Plett, *Radio Frequency Integrated Circuit Design*. Norwood, MA: Artech House, 2003.

[3] K. S. Kundert, J. K. White, and A. Sangiovanni-Vincentelli, *Steady-State Methods for Simulating Analog and Microwave Circuits*. Boston, MA: Kluwer, 1990.

[4] M. S. Nakhla and J. Vlach, "A piecewise harmonic-balance technique for determination of periodic response of nonlinear systems," *IEEE Trans. Circuits Syst.*, vol. 23, no. 2, pp. 85–91, Feb. 1976.

[5] L. O. Chua and P. M. Lin, *Computer-Aided Analysis of Electronic Circuits*. Englewood Cliffs, NJ: Prentice-Hall, 1975.

[6] S. Maas, *Nonlinear Microwave and RF Circuits*. Upper Saddle River, NJ: Artech House, 2003.

[7] D. Tannir and R. Khazaka, "Moments based computation of intermodulation distortion in RF circuits," *IEEE Trans. Microwave Theory Tech.*, vol. 55, no. 10, pp. 2135–2146, Oct. 2007.

[8] E. Gad, R. Khazaka, M. Nakhla, and R. Griffith, "A circuit reduction technique for finding the steady-state solution of nonlinear circuits," *IEEE Trans. Microwave Theory Tech.*, vol. 48, no. 12, pp. 2389–2396, Dec. 2000.

[9] D. Tannir and R. Khazaka, "Adjoint sensitivity analysis of nonlinear distortion in RF circuits," in *Proc. Custom Integr. Circuits Conf.*, Sep. 2009, pp. 633–636.

[10] C. W. Ho, A. E. Ruehli, and P. A. Brennan, "The modified nodal approach to network analysis," *IEEE Trans. Circuits Syst.*, vol. 22, no. 6, pp. 504–509, Jun. 1975.

[11] J. W. Bandler, Q. J. Zhang, and R. M. Biernacki, "A unified theory for frequency-domain simulation and sensitivity analysis of linear and nonlinear circuits," *IEEE Trans. Microwave Theory Tech.*, vol. 36, no. 12, pp. 1661–1669, Dec. 1988.

[12] R. Griffith and M. Nakhla, "A new high order absolutely stable explicit numerical integration algorithm for the time domain simulation of nonlinear circuits," in *Proc. ACM ICCAD*, 1997, pp. 504–509.

[13] J. Vlach and K. Singhal, *Computer Methods for Circuit Analysis and Design*. New York: Van Norstrand, 1983.

[14] S. K. Mitra, *Digital Signal Processing: A Computer-Based Approach*. New York: McGraw-Hill, 2005.

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 behavior synthesis framework which can efficiently extract similar structures in programs. Our approach is very scalable in benefit 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**

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; huihuang@cs.ucla.edu; wjiang@cs.ucla.edu).

Digital Object Identifier 10.1109/TCAD.2011.2106370

7% latency overhead with our pattern techniques on the Xilinx Virtex-4 field-programmable gate arrays, compared to the traditional behavioral synthesis flow.

***Index Terms*—Area optimization, behavioral synthesis, edit distance, pattern mining.**

I. Introduction

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 has 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 breath-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 by 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

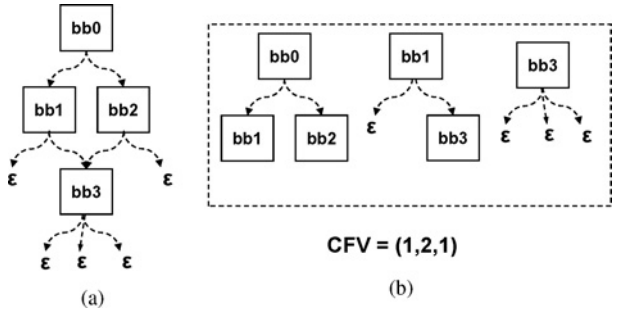

Fig. 1. (a) CFG. (b) Control flow features and CV for the given CFG, assuming bb1 and bb2 are similar supernodes.

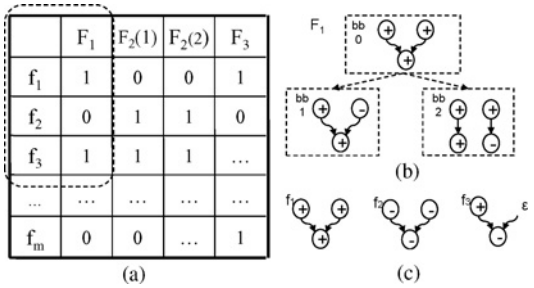

Fig. 2. (a) Sample *feature map matrix*. (b) Sample CFG feature F_1. (c) Three sample DFG features

labeled graph of a DFG by choosing the label of a node to be 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, while edges of a− operation should have different 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 CFG 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 *CFG feature* is a subgraph $S = \{u,\ l_1, ...,\ l_m \mid u,\ l_i \in V_{bb}\} \subseteq G$, such that edge $(u,\ l_i) \in E_G$ (m equals the number of outputs for supernode u).

Fig. 1 shows the features of a CFG graph, as well as the corresponding CV. Here assume basic blocks 1 and 2 are similar, therefore, the number of the second feature in CFV 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 ($\sum_{i=1}^{n} C_i$) 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 $C\_INSERT_i = C\_DETELE_i = w_i$, and the replacement cost is $C\_REPLACE_{i,j} = |w_i - w_j|$. 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, $\|CV(G_1) - CV(G_2)\|_1 \leq 4 * wd(G_1, G_2)$ ($\|X\|_1 = \sum |x_i|$ is L_1 norm).

Proof: One edit operation op_i can at most change CV in terms of l_1 norm by 4. And since C_i is a positive integer (i.e., $C_i \geq 1$), we have $\|CV(G_1) - CV(G_2)\|_1 \leq 4 \leq 4 * C_i$. Adding all the edit operations together, we come to the conclusion that $\|CV(G_1) - CV(G_2)\|_1 \leq 4 * \sum C_i = 4 * wd(G_1, G_2)$. ■

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 CFG 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 CFG feature. Each entry records whether a DFG feature appears in a target CFG feature. As shown in Fig. 2, DFG features f_1 and f_3 appear once in CFG 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_{missUb} is defined to be the maximum number of columns covered by $4 * l_{dist}$ rows in feature map matrix.

Theorem 2: Let l_{dist} be the given edit distance limit, and $CV_{CFG}(G_i)$ be the *control flow CV* of G_i. If the generalized edit distance (GED) between G_1 and G_2 does not exceed l_{dist}, we have $\|CV_{CFG}(G_1) - CV_{CFG}(G_2)\|_1 \leq l_{missUb}$.

Proof: Assume the GED between G_1 and G_2 is less than l_{dist}, Theorem 1 tells us that the *data flow CV* distance between G_1 and G_2 is no more than $4*l_{dist}$; 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 $\|CV_{CFG}(G_1) - CV_{CFG}(G_2)\|_1$ cannot exceed l_{missUb}. ■

Based on the analysis above, we know that subgraphs with CV difference larger than l_{missUb} 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 two 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, in which $SF_i = \{fg_{i1}, fg_{i2}, ..., fg_{iN}\}$ and fg_{ij} is the jth subgraph fragment in set i. Assume the edit distance between graph fg_{1a} and fg_{2b} is $d(fg_{1a}, fg_{2b})$, the GED between SF_1 and SF_2 is defined as $\min \sum_{i=1}^{N} d(fg_{1i}, fg_{2p_i})$, 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 ith subgraph fragment in set 1 and the jth fragment in set 2. When we build M, the condition edit distance $d(fg_{1i}, fg_{2j}) \leq l_{dist}$ 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 ith fragment in the first set and p_ith 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 relabeling 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:

$$\frac{N * mux(io) + area(P)}{N * (mux(io) + mux(internal)) + area(P)} + \alpha * \frac{latency(P)}{|P|}. \quad (1)$$

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 $\{P_0|1\ ;\ 2\}$, $\{P_1|0\ 1\ ;\ 1\ 3\ ;\ 4\ 5\}$, and $\{P_2|3\ 4\ ;\ 5\ 6\}$. 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 $F(p_0, ..., p_3)$ 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 $f_0 = (\neg p_0 + \neg p_1) \cdot (\neg p_0 + \neg p_2)$

TABLE II
RESOURCE REDUCTION ON ALL TEST CASES WITH CDFG PATTERN RECOGNITION

	FF (np)	LE (np)	FF (DFG)	CMP (%)	LE (DFG)	CMP (%)	FF (CDFG)	CMP (%)	LE (CDFG)	CMP (%)
IDCT	1514	5071	1601	5.75	3998	−21.15	1193	−21.20	2870	−43.40
SYNFILT	476	1578	394	−17.22	1193	−24.40	325	−31.72	1070	−32.19
BLKSORT	295	1565	277	−6.10	1455	−7.03	235	−20.34	1147	−26.71
BH	850	3288	701	−17.52	2692	−18.13	640	−24.71	2659	−19.13
HEAP	1023	6743	945	−7.62	6291	−6.70	934	−8.61	5995	−11.09
LEXTREE	824	4211	714	−13.35	3740	−11.18	695	−15.66	3543	−15.86
Average				−9.34		−14.76		−20.37		−24.73

TABLE I
CDFG PATTERN RECOGNITION RESULTS

	Line	Pattern	Inst	Avg. Calc	MAX
IDCT	215	5	10	1.02	64
SYNFILT	1051	3	7	0.94	24
BH	301	6	14	1.33	37
BLKSORT	289	3	6	0.82	15
HEAP	217	11	23	1.49	10
LEXTREE	696	6	12	1.31	24

to 1. Based on the discussion above, the final CNF constraint is $F(p_0, p_1, p_2, p_3) = f_0 f_1 f_2 f_3 = 1$, 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 flow has been applied to six real-life test cases containing common control flow structures in the program. Our test cases include IDCT, SYNFLIT, 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.

TABLE III
PATTERN RECOGNITION RESULTS ON ALL C TEST BENCHMARKS IN SPEC2006

	No. of Lines	Patterns	Instances	MAX	Time (s)
401.bzip2	8300	505	986	68	118.2
429.mcf	2692	57	61	9	61.0
433.milc	15 049	455	1745	33	262.5
445.gobmk	197 215	6080	50 969	25	1293.2
456.hmmer	35 999	1277	3571	38	700.2
458.sjeng	13 854	1559	3250	21	495.5
462.libquantum	4364	57	248	20	19.0
470.lbm	1162	46	123	41	66.4
482.sphinx3	23 380	685	2648	50	297.1
998.specrand	81	1	2	13	9.4

The pattern-based FPGA resource reduction results with our CDFG pattern recognition algorithm applied are shown in Table II. Our letter has been compared to a traditional behavioral synthesis flow without pattern optimization technique involved and the one with DFG pattern recognition. In Table II, the second, third, and fifth columns show the synthesis results for the number of registers used without pattern-based technique, with a DFG pattern-based technique and with CDFG pattern-based technique, respectively. Columns 7–11 list the amount and comparison of logic elements usage in those benchmarks.

Overall, our CDFG pattern synthesis flow has a 24% resource reduction on average compared with the traditional one, and outperforms the one with data flow pattern only for most of the test cases. 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]. The runtime and discovered patterns are shown in Table III.

Table III shows that our pattern recognition can effectively discover very large patterns from complex programs. The first column contains names of benchmarks; for each row,

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

[1] L. M. Wills, "Automated program recognition by graph parsing," Ph.D. dissertation, Dept. Comput. Sci., Massachusetts Inst. Technol., Cambridge, MA, 1992.

[2] B. D. Martino and G. Iannello, "PAP recognizer: A tool for automatic recognition of parallelizable patterns," in *Proc. IWPC*, 2004, pp. 164–174.

[3] C. Alias, "Program optimization by template recognition and replacement," Ph.D. dissertation, Dept. Inform. Syst., Univ. Versailles, Versailles, France, 2005.

[4] C. Keler, "Pattern-driven automatic parallelization," *Sci. Program.*, vol. 5, no. 3, pp. 251–274, 1996.

[5] R. Metzger and Z. Wen, *Automatic Algorithm Recognition and Replacement: A New Approach to Program Optimization*. Cambridge, MA: MIT Press, 2000.

[6] M. A. Abdulrahim and M. Misra, "A graph isomorphism algorithm for object recognition," *Patt. Anal. Applicat.*, vol. 1, no. 3, pp. 189–201, 1998.

[7] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*. San Diego, CA: Academic Press, 1999.

[8] K. Keutzer, "DAGON: Technology binding and local optimization by DAG matching," in *Proc. 24th Des. Automat. Conf.*, 1987, pp. 341–347.

[9] D. Rao and F. J. Kurdahi, "On clustering for maximal regularity extraction," *IEEE Trans. Comput.-Aided Des.*, vol. 12, no. 8, pp. 1198–1208, Aug. 1993.

[10] P. Brisk, A. Kaplan, R. Kastner, and M. Sarrafzadeh, "Instruction generation and regularity extraction for reconfigurable processors," in *Proc. CASES*, 2002, pp. 262–269.

[11] M. R. Corazao, M. A. Khalaf, L. M. Guerra, M. Potkonjak, and J. M. Rabaey, "Performance optimization using template mapping for datapath-intensive high-level synthesis," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 15, no. 8, pp. 877–888, Aug. 1996.

[12] O. Bringmann and W. Rosenstiel, "Resource sharing in hierarchical synthesis," in *Proc. IEEE/ACM ICCAD*, Nov. 1997, pp. 318–325.

[13] T. Ly, D. Knapp, R. Miller, and D. MacMillen, "Scheduling using behavioral templates," in *Proc. 32nd ACM/IEEE Conf. DAC*, Jun. 1995, pp. 101–106.

[14] B. T. Messmer and H. Bunke, "A new algorithm for error-tolerant subgraph isomorphism detection," *IEEE Trans. Patt. Anal. Mach. Intell.*, vol. 20, no. 5, pp. 493–504, May 1998.

[15] J. Huan, W. Wang, and J. Prins, "Efficient mining of frequent subgraphs in the presence of isomorphism," in *Proc. ICDM*, 2003, pp. 549–552.

[16] M. Kuramochi and G. Karypis, "Frequent subgraph discovery," in *Proc. ICDM*, 2001, pp. 313–320.

[17] X. Yan and J. Han, "gSpan: Graph-based substructure pattern mining," in *Proc. ICDM*, 2002, pp. 721–724.

[18] A. Inokuchi, T. Washio, and H. Motoda, "An a priori-based algorithm for mining frequent substructures from graph data," in *Proc. PKDD*, 2000, pp. 13–23.

[19] J. Cong and W. Jiang, "Pattern-based behavior synthesis for FPGA resource reduction," in *Proc. 16th Int. ACM/SIGDA Symp. FPGA*, 2008, pp. 107–116.

[20] C. Douligeris and G. Feng, "Using Hopfield networks to solve assignment problem and n-queen problem: An application of guided trial and error technique," in *Proc. 2nd Hellenic Conf. AI SETN* , 2002, pp. 325–336.

[21] X. Li, M. F. Stallmann, and F. Brglez, "Effective bounding techniques for solving unate and binate covering problems," in *Proc. 42nd Annu. DAC*, 2005, pp. 385–390.

[22] J. Cong, Y. Fan, G. Han, W. Jiang, and Z. Zhang, "Platform-based behavior-level and system-level synthesis," in *Proc. IEEE SOCC*, Sep. 2006, pp. 199–202.

[23] *GMT Toolkit* [Online]. http://www.cs.sunysb.edu/algorith/implement/gmt/implement.shtml

[24] *Xilinx, Inc.* [Online]. Available: http://www.xilinx.com

[25] *SPEC CPU2006* [Online]. Available: http://pec.it.miami.edu/cpu2006