Fast Filter-Based Boolean Matchers

Chaofan Yu, Lingli Wang, Chun Zhang, Yu Hu, and Lei He

***Abstract*—Boolean matching is one of the fundamental and time-consuming procedures in field-programmable gate array (FPGA) synthesis. The SAT-based Boolean matchers (BMs) are not scalable while other Boolean matchers based on complicated Boolean logic operation algorithms are not flexible for complex PLBs. Recently, a scalable Boolean matcher (F-BM) based on the Bloom filter has been proposed for both scalability and flexibility. However, it requires large amount of memory space which can be a bottleneck for traditional personal computers. To tackle that problem, this letter proposes a novel Boolean matcher with much less memory requirement. Compared with F-BM, the proposed Boolean matcher has achieved an average of 5% better result with 2000x smaller storage and only 1.6x more runtime when applying to the same application. The significant reduction of storage requirements makes the proposed Boolean matcher able to handle more complicated PLB structures with larger input sizes.**

***Index Terms*—Boolean matching, field-programmable gate array (FPGA), NPN, resynthesis, SAT.**

I. Introduction

DURING the process of binding a logic network to a set of cells, a fundamental problem is recognizing whether a function (e.g., part of the logic network) can be implemented by one or several cells.

Boolean matching is a technique to detect whether such binding can be accomplished or not. In field programmable gate array (FPGA), such binding become more complicated due to the configurability of the cell, for example, the programmable logic blocks (PLBs) consists of look-up tables (LUTs) and macro gates together. For a certain PLB structure, Boolean matching algorithms based on function decomposition [9][10] are favorable for the efficiency and low storage requirement. However, when applied to a different PLB structure the algorithms must be carefully modified or even entirely developed from scratch, thus losing the flexibility. Another class of Boolean matching algorithms based on canonicity and Boolean signatures maintain the scalability but can only be applied on functions of limited sizes [6], [7].

Manuscript received July 07, 2013; accepted August 24, 2013. Date of publication September 05, 2013; date of current version November 20, 2013. This work was supported in part by the National Natural Science Foundation of China (under Grants 61272070, 61131001, and 61171011) and a research gift from Cisco. This manuscript was recommended for publication by Z. Shao.

C. Yu and L. Wang are with the State Key Lab of ASICs, Fudan University, Shanghai 201203, China (e-mail: 11212020062@fudan.edu.cn, llwang@fudan.edu.cn).

C. Zhang is with the Electrical Engineering Department, Missouri University of Science and Technology, Columbia, MO 65409 USA (e-mail: zhanchun@mst.edu).

Y. Hu is with the School of Optical and Electronic Information, Huazhong University of Science and Technology, Wuhan 430000, China (e-mail: huyu.cs@gmail.com).

H. Lei is with the Electrical Engineering Department, University of California, Los Angeles, CA 90024 USA (e-mail: lhe@ee.ucla.edu).

Digital Object Identifier 10.1109/LES.2013.2280582

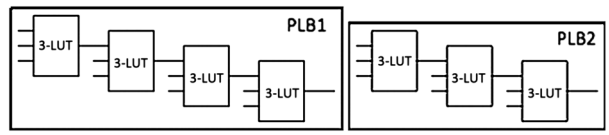

Fig. 1 PLBs to match.

As the number of new FPGA devices growing up, an ideal Boolean matcher which can be used for various kinds of PLB architectures is desired. Since all Boolean matching problems can be transformed to SAT problems, SAT based Boolean matchers (SAT-BMs) have been proposed [4][15]. Due to the intrinsic computational complexity of the SAT problem, SAT-BMs have poor scalability to complex PLB structures. Given the fact that an implementable function (SAT) is much faster than that for a nonimplementable function (UNSAT), an efficient BM based on Bloom filter (called F-BM) has been proposed in [8][14], which are proved to have 80x speed up and only 11% loss of area reduction compared with existing improved SAT-BMs when applying in area-oriented resynthesis as an example. The F-BM has further been applied to the recently proposed software-as-a-service (SaaS) based Boolean matching platform [1].

One major problem in F-BM is that the Bloom filter built for a 9-input structure (*PLB1* in Fig. 1) and a 7-input structure (*PLB2* in Fig. 1) consumes large memory space (e.g., 2 GB) when running on a traditional PC. This memory requirement also increases with the number of inputs of the PLB to be matched. The second problem is the time consumed in building the Bloom filter library during the training process, which can take up to 1 week. This is because all functions to be trained needs to be verified using the SAT-solver. This potentially wastes large amount of time, since many functions trained are actually NPN (Negation–Permutation–Negation) equivalent. To control the library size and to reduce the training time, F-BM sets up the upper limit for number of functions to be trained, which may degrade the training quality since many functions are not able to be trained.

In a word, the huge memory consumption has limited the applicability of F-BM. One solution is to increase the memory resources which are implemented in [1] as SaaS Boolean Matcher. Another method is to reduce the storage, which is the main work of this letter. Besides, after the storage occupation is reduced, we can use hash table to store SAT results directly instead of resolving SAT problem at runtime.

In this letter, we propose two Boolean matchers. FC-BM (Filter-based BM with canonical form) has reduced the storage of F-BM by 2000x by introducing a semicanonical form as an intermediate procedure while still using high space efficiency lookup table Bloom filter. The other one, FH-BM (Filter-Hash-based BM), stores all the implementable truth tables and the corresponding configuration in the hash table to avoid performing SAT computation at runtime, which results in significant speedup. Furthermore as the permutations are

considered in the classification, the time consumed in building library is greatly reduced. Compared with F-BM, when applied to the same resynthesis process, the FC-BM achieves a better result with an average of 5% more area reduction, 2000x smaller storage requirements, with only 1.6x more runtime; the FH-BM achieves an average of 5% more area reduction, 40x smaller storage requirements, with 20x speedup.

The remaining of this letter is organized as follows. Section II introduces terminologies. Section III describes the proposed new Boolean matchers. Section IV presents experimental results. Section V concludes the letter.

II. Terminologies

A. Boolean Matching

Given a PLB structure with programmable logic units and macro-gates which can have up to P input pins, we can write this PLB as $H(P)$. The programmable unit is often implemented as LUTs. A K-LUT means a cell which can implement any function $f(X)$ with $|X| \leq K$ by storing the all 2^K truth table values. Considering a Boolean logic function $f(X)$ with $|X| \leq |P|$, Boolean matching is used to confirm the possibility of binding this function to the PLB $H(P)$.

For macro-gates such as AND or XOR gates, the possibility is easy to confirm. For a K-LUT with a function $f(X)$ where $|X| \leq K$, the answer is also confirmed. However, when combining macro-gates & LUTs together, the question becomes nontrivial. Given a certain PLB structure, if a function is implementable, we call this function is SAT, otherwise UNSAT.

B. NPN Canonical Form

Given two functions, NPN equivalence between them is obtained when it is possible to achieve identical values for both truth table outputs by permutation and/or negation of the function inputs and/or negation of the function output [3]. The NPN canonical form is defined based on a property that makes it unique among all functions in an NPN-equivalence class [3]. The key task is to devise a canonical form that handles permutation and complementation of inputs and output with a low average time complexity. If a PLB can implement a function $f(X)$, then all other functions which are NPN equivalent with $f(X)$ can be also implemented by this PLB if it supports complementation of inputs and output, where most PLBs do.

C. Bloom Filter

When asking if one element exists in the collection of them, the favored data structure could be hash table. Compared with hash table, Bloom filter is also a data structure which helps judging the existence of one element, however, with little fault possibility in the judgment. If one element does exist, the answer would be always correct but when the element does not, there is a little possibility that the Bloom filter would give the wrong answer.

Bloom filter is a bit array with m bits. When adding an element, it sets k bits of the array to 1. The positions of k bits are calculated by k different hash functions. However, remove element is not allowed for it may break the data structure when some other elements share some bits with this one. To query an element, it just tests whether the k bits are all set to 1 [11].

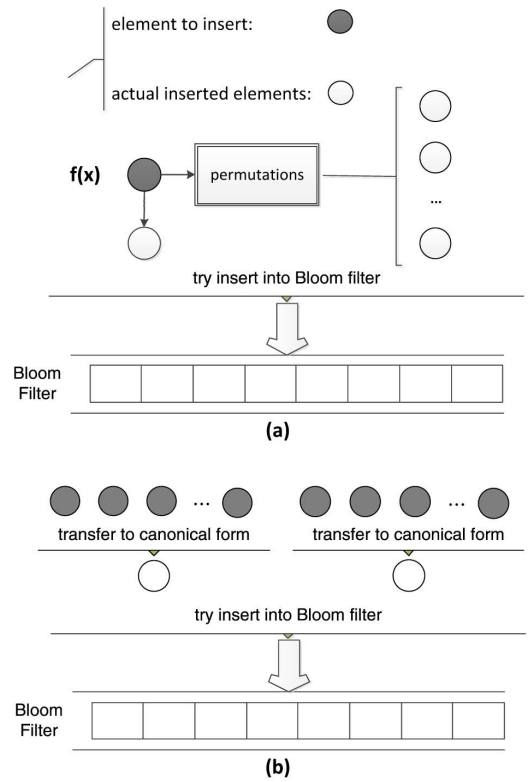

Fig. 2 Building Bloom filter (a) F-BM (b) for FC-BM.

Bloom filter is very space-efficient because the size of set has no relationship with the size of element itself or the number of elements to be added. However, the increasing number of elements would result in a higher query fault possibility.

III. Proposed Boolean Matchers

This section presents the details of FC-BM and FH-BM. The FC-BM focuses on reducing the space complexity while FH-BM is to reduce the time complexity.

To make the comparison persuasive, the PLBs to match (shown in Fig. 1) and the truth tables to satisfy in our new Boolean matchers are the same as in F-BM.

A. Reduce Bloom Filter Size

Firstly, let's focus on the size of Bloom filter in FC-BM and the way to perform query in FC-BM.

Fig. 2 describes the difference of building Bloom filter procedure between F-BM and FC-BM. In F-BM, for every truth table T which is implementable, we insert not only the truth table itself but also 10,000 of its permutations into the Bloom filter in the F-BM. However, in FC-BM, T is transformed into canonical form first before inserting into Bloom filter, which results in significantly decreasing on the Bloom filter size given the NPN equivalence class size of T can reach up to $N!2^{N+1}$ if the input size is N.

As the Bloom filter only stores canonical form of SAT truth tables, the process of querying element existence needs one more canonization step. Obviously, the speed of getting canonical form is important for it is executed every time before querying.

TABLE I
SEMI-CANONICAL FUNCTION NUMBER

Input size	Practical function number	Semi-canonical function number	ratio
5	722369	2482	0.34%
6	1814248	14252	0.79%
7	4753232	71697	1.51%
8	12606209	291244	2.31%
9	17849186	1065756	5.97%
total	37745244	1445431	3.83%

B. Improvements of BM Results

In F-BM, 10,000 limited permutations are stored given the space complexity, which leads to a portion of NPN equivalence class missing, especially when the truth table size grows up. In FC-BM, such missing can be avoided by introducing the canonical form which helps discovering all NPN equivalence class, thus improving Boolean matching results.

C. Canonical Form Selection

This letter chooses a semicanonical form rather than using exact canonical form. Although using exact canonical algorithm could achieve a better result in reducing Bloom filter size, the computation complexity of exact NPN canonical form would result in the increasing of running time. The semicanonical algorithm used in this letter is from [5], which has been proved to have good performance in Lazy Man's Synthesis [5].

Using the semicanonical form has been proved to reduce the Bloom filter size by several orders of magnitude, which is quite acceptable in our improvements.

Table I show the result of semicanonical form of 5-9 input practical functions used in F-BM. The number of practical functions trained in [8] is about 37 000 000, after calculating permutations for each practical function (as shown in Fig. 2 (a)), the actually stored functions number is about 3 billion in F-BM. By applying semicanonical algorithm to these practical functions, the actually stored functions number in our new Boolean matcher is about 1 400 000, which results in a decreasing of 2000x times in the Bloom filter size. The average training time (judging SAT or UNSAT) is reduced about 25x times for now only semicanonical functions need to be trained.

D. New Boolean Matcher: FC-BM

The new Boolean matcher FC-BM is similar to F-BM while it introduces the semicanonical form in truth table querying and storing. As Bloom filter size has been reduced significantly in FC-BM, it is expected to support much more PLBs instead of only 2 PLBs in Fig. 1, which makes it possible to implement a practical Boolean matcher on a local PC. Also the speed of FC-BM should be promoted due to the reduced memory IO operations.

However, the introduced canonical form might cause the FC-BM running slower than F-BM. The reason is that FC-BM can discover more functions which are expected to be SAT than F-BM since F-BM only considers 10,000 limited permutations. For those additional discovered functions in FC-BM, whether they are actually SAT or not, FC-BM always spends time performing SAT computation, thus requiring more runtime especially when the function is actually UNSAT.

E. New Boolean Matcher: FH-BM

Bloom filter is adopted in F-BM for its great space efficiency. Now that the space complexity is decreased significantly by introducing the semicanonical form, the truth tables can be stored in other data structures instead of Bloom filter. This section introduces another Boolean matcher FH-BM, which is based on hash table.

There are two reasons contribute to the speed up of FH-BM. The dominate reason is that FH-BM avoids performing SAT computation at runtime by storing the SAT results in the hash table. The other reason lies in the false positive rate of the Bloom filter. The truth tables which are actually UNSAT may be judged to SAT in F-BM, thus increasing unnecessary computation. Besides, the UNSAT truth tables often need more time [8][14].

The limitation of FH-BM is that the library size is very sensitive to the input size for the truth tables. In FH-BM, every N input function takes up bits. For PLBs which accepts small input (less than 14) functions, the FH-BM will be much more favored than F-BM considering the great speedup.

IV. RE-SYNTHESIS USING FC-BM AND FH-BM

The application, area-oriented resynthesis described in [2], is adopted to show the effectiveness of the proposed FC-BM and FH-BM against F-BM. The test input of resynthesis is a circuit mapped to 3-LUTs (mapped by ABC [12]). The resynthesis scans the combinational portion of the circuit in a topological order and generating new logic blocks by combining the logic blocks at the input LUTs. Each logic block is mapped against *PLB1* and *PLB2* (shown in Fig. 1). When a mapping is found by the Boolean matcher, the logic block is replaced by the corresponding PLB structure if such a replacement reduces the number of LUTs used to implement this logic block. The algorithm terminates when all LUTs are scanned once and no further replacements are performed.

The pseudo-codes of resynthesis flow are shown in Fig. 3 and Fig. 4 respectively. In the experiments, we compared F-BM, FC-BM and FH-BM with SAT-BM, respectively. Fifteen circuits from 3 independent benchmark sets (MCNC, industrial designs and IWLS 2005 [13]) are tested. The experiments are running on a Linux server (CPU: Intel Xeon Processor X5650 with 12M cache, Memory: 64 GB).

Table II shows the reduced LUTs of each BM. As a result, F-BM loses about 11% reduced LUTs where our BMs (FC-BM and FH-BM) have only 6% loss compared with SAT-BM.

Table III compares the runtime of each BM. In comparison with SAT-BM, FH-BM achieves a 1729x speedup, which is about 20x times faster than F-BM; FC-BM achieves a 48x speedup, which loses about 40% speedup against F-BM's 80x speedup.

Table IV shows the differences of the five Boolean matchers in this experiment, where we can see that FH-BM is even faster than SaaS-BM with better results. However as discussed above, the storage requirement will be the bottle neck of FH-BM when the input size grows up or matching more structures in local PC, just as F-BM does. For a local PC, the FC-BM would be more favored for it supports 2000x more trained functions than F-BM with acceptable speedup, thus enabling FC-BM to support more structures rather than 2 PLBs and accept larger input size.

```
ResynthesisByFC-BM(network, parameters)

  for each node of network in topological order
    cutset = enumerateKfeasibleCut(node, parameters);
    for each cut in cutset
      for each PLB H in the PLB libraries
        if(|cut| >= |H|) continue; // No area reduction
        canonicalcut = getCanonicizeForm(cut)
        if(canonicalcut exists in Bloom Filter) // filter
          if(SAT-BM (cut, H, conf) == SAT) // SAT
            setconf(H, conf) // configure PLB
            updateNetwork(cut, H); // replace
```

Fig. 3 Pseudocode of resynthesis based on FC-BM.

```
ResynthesisByFH-BM(network, parameters)

  for each node of network in topological order
    cutset = enumerateKfeasibleCut(node, parameters);
    for each cut in cutset
      for each PLB H in the PLB libraries
        if(|cut| >= |H|) continue; // No area reduction
        canonicalcut = getCanonicalForm(cut)
        if(canonicalcut exists in Hash Table) // SAT
          // get conf directly from hash table
          canoconf = getPLBConf(canonicalcut, H)
          conf= revertConf(cut, canonicalcut, canoconf, H)
          setconf(H, conf) // configure PLB
          updateNetwork(cut, H); // replace
```

Fig. 4 Pseudocode of resynthesis based on FH-BM.

TABLE II
EXPERIMENTS RESULTS OF REDUCED LUT

		Reduced LUT#						
		SAT-BM	F-BM	ratio	FC-BM	ratio	FH-BM	ratio
MCNC	alu4	14	14	100%	14	100%	14	100%
	diffeq	4	3	75%	3	75%	3	75%
	ex5p	1	1	100%	1	100%	1	100%
	s298	7	7	100%	7	100%	7	100%
	seq	1	1	100%	1	100%	1	100%
Industrial	Ex1	1721	1139	66%	1537	89%	1537	89%
	Ex2	305	271	89%	291	95%	291	95%
	Ex3	674	549	81%	607	90%	607	90%
	Ex4	672	513	77%	585	87%	585	87%
	Ex5	93	79	85%	88	95%	88	95%
IWLS	leon2	6673	6456	97%	6609	99%	6609	99%
	leon3	10770	10395	97%	10602	98%	10602	98%
	leon3mp	6941	6636	96%	6802	98%	6802	98%
	netcard	5566	5435	98%	5509	99%	5509	99%
	Geomean	-	-	89%	-	94%	-	94%

TABLE III
EXPERIMENTS RESULTS OF RUNTIME

		Runtimes(s)						
		SAT-BM	F-BM	ratio	FC-BM	ratio	FH-BM	ratio
MCNC	alu4	246	7.86	31	6.78	36	0.047	5234
	diffeq	392	0.05	7840	0.26	1508	0.007	56000
	ex5p	0.02	0.02	1	0.02	1	0.005	4
	s298	8.89	0.08	111	0.17	52	0.028	318
	seq	3.93	0.02	197	0.02	197	0.004	983
Industrial	Ex1	11892	101	118	153	78	2.238	5314
	Ex2	1587	26	61	95	17	0.416	3815
	Ex3	32156	24	1340	77	418	0.566	56813
	Ex4	10280	86	120	475	22	1.851	5554
	Ex5	671	6.66	101	8.9	75	0.099	6778
IWLS	leon2	63834	1771	36	1678	38	87	734
	leon3	60959	1976	31	2198	28	114	535
	leon3mp	46063	1382	33	1495	31	53	869
	netcard	23626	1342	18	1119	21	60	394
	Geomean		-	80	-	48	-	1729

TABLE IV
OVERVIEW OF COSTS IN RESYNTHESIS FOR PLB1 & PLB2 IN FIG. 1

	speedup	library size	area reduction	training time
SAT-BM	1x	no need	1	no need
F-BM	80x	2G local	0.89	1
FC-BM	48x	1M local	0.94	0.04
FH-BM	1729x	60M local	0.94	0.04
SaaS-BM [1]	863x	>2G remote	0.89	1

V. CONCLUSION

In this letter, we have presented two new Boolean matchers: the FC-BM and FH-BM. FC-BM is very high space efficient for it combines the merits of both Bloom filter and NPN-equivalence. When applying FC-BM on the same resynthesis application, FC-BM has reduced the training time about 25x times, Bloom filter size about 2000x times. The significant library size reduction makes FC-BM applicable on more structures instead of 2 and larger input size rather than 9 on a traditional PC. Besides, FC-BM has discovered more SAT functions than F-BM, which results in an improvement of 5% more area reduction. However it's 40% slower than F-BM. FH-BM is very fast for it avoids the SAT calculation, but it is only favored under limited structures and input sizes.

REFERENCES

[1] C. Zhang, Y. Hu, L. Wang, L. He, and J. Tong, "Engineering a scalable Boolean matching based on EDA SaaS 2.0," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, , 2010, pp. 750–755.

[2] A. Ling, D. Singh, and S. Brown, "FPGA technology mapping: A study of optimality," in *Proc. Design Autom. Conf.*, , 2005, pp. 427–432.

[3] D. Chai and A. Kuehlmann, "Building a Better Boolean Matcher and Symmetry Detector," in *Proc. Design Autom. Conf. Eur. (DATE)*, , 2006, pp. 1–6.

[4] Y. Hu, V. Shih, R. Majumdar, and L. He, "Exploiting symmetry in SAT-based Boolean matching for heterogeneous FPGA technology mapping," *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, pp. 350–353, 2007.

[5] W. Yang, L. Wang, and A. Mishchenko, "Lazy Man's Synthesis," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, , 2012, pp. 597–604.

[6] A. Abdollahi and M. Pedram, "A new canonical form for fast Boolean matching in logic synthesis and verification," in *Proc. Design Autom. Conf.*, .

[7] L. Benini and D. Micheli, "A survey of Boolean matching techniques for library binding," *ACM Trans. Design Autom. Electron. Syst.*, vol. 2, no. 3, pp. 193–226, Jul. 1997.

[8] C. Zhang, Y. Hu, L. Wang, L. He, and J. Tong, "Building A Faster Boolean Matcher Using Bloom Filter," in *Proc. 18th Annu. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays* , , 2010, pp. 185–188.

[9] J. Cong and Y.-Y. Hwang, "Boolean matching for LUT-based logic blocks with applications to architecture evaluation and technology mapping," *J. Technol. Comput.-Aided Design* , vol. 20, no. 9, pp. 1077–1090, Sep. 2001.

[10] A. Mishchenko, R. K. Brayton, and S. Chatterjee, "Boolean factoring and decomposition of logic networks," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, , 2008, pp. 38–44.

[11] Bloom filter [Online]. Available: http://en.wikipedia.org/wiki/Bloom_filter

[12] ABC: A System for Sequential Synthesis and Verification [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/abc/

[13] IWLS 2005Benchmarks [Online]. Available: http://iwls.org/iwls2005/benchmarks.html

[14] C. Zhang, Y. Hu, L. Wang, L. He, and J. Tong, "Accelerating Boolean Matching Using Bloom Filter," *IEICE Trans. Fundamentals Electron., Commu. Comput. Sci.*, vol. E93-A, no. 10, pp. 1775–1781, Oct. 2010.

[15] S. Safarpour, A. Veneris, G. Baeckler, and R. Yuan, "Efficient SAT-based Boolean matching for FPGA technology mapping," in *Proc. Design Autom. Conf.*, , 2006, pp. 466–471.