

# Exploiting Symmetry in SAT-Based Boolean Matching for Heterogeneous FPGA Technology Mapping

Yu Hu<sup>1</sup>, Victor Shih<sup>2</sup>, Rupak Majumdar<sup>2</sup> and Lei He<sup>1</sup>  
1. Electrical Engineering Department  
2. Computer Science Department  
University of California, Los Angeles \*

## ABSTRACT

The Boolean matching problem is a key procedure in technology mapping for heterogeneous Field Programmable Gate Arrays (FPGA), and SAT-based Boolean matching (SAT-BM) provides a highly flexible solution for various FPGA architectures. However, the computational complexity of state-of-the-art SAT-BM prohibits its application practically. In this paper we propose an efficient SAT-BM algorithm by exploring function and architectural symmetries. While the most recent work obtained up to 13x speedup, we achieve up to 200x speedup, when both are compared to the original SAT-BM algorithm.

## 1. INTRODUCTION

FPGAs are programmable logic chips that can be configured to implement various digital circuits. FPGAs are quickly replacing custom ASICs in many areas due to their flexibility and fast turnaround times for product development. However, these benefits come at the heavy cost of area, speed, and power.

The programmable logic block (PLB) is the basic element of an FPGA design. Various programmable devices can be placed within a PLB; a lookup table (LUT) is one such programmable device. LUT-based FPGAs use PLBs populated with LUT components to implement various logic functions. A K-LUT device consists of  $K$  inputs, one output, and  $2^K$  configuration bits that serve as truth table entries. With its  $2^K$  configuration bits programmed accordingly, the K-LUT can implement any  $K$ -input function.

Given a logic-level design, a crucial step in the overall FPGA computer-aided design (CAD) flow is *technology mapping*. This step converts a circuit into a network of PLBs. The circuit function can be given in terms of a synthesized multi-level netlist, input/output functional relationship, or other representation. Depending on the technology mapping approach, the resulting network will exhibit direct area, delay, and power costs. Most of the existing work for FPGA technology mapping [1, 2, 3] assumes that the only logic elements within a PLB are K-LUTs and the resulting FPGA is *homogeneous*. Since a K-LUT can implement any  $K$ -input function, the goal of technology mapping for homogeneous FPGAs is to find optimal  $K$ -bounded covers [3] in the subject graph; the logic functionality of each  $K$ -bounded cover does not need to be considered.

Alternatively, modern FPGAs such as Xilinx Virtex IV [4] and Altera Stratix II [5] employ *heterogeneous* PLBs, which contain various logic devices such as logic gates and LUTs with different inputs. This heterogeneity allows more flexibility in FPGA designs,

which can reduce power dissipation and area, and improve performance. On the other hand, the extra flexibility of heterogeneous FPGAs increases the search space of technology mapping. As an example, suppose we map a design to an FPGA with  $K$ -input heterogeneous PLBs; the functionality of each  $K$ -bounded cover must be considered explicitly during technology mapping.

*Boolean matching* [6, 7] is the most important sub-problem in technology mapping for heterogeneous FPGAs. Given a target FPGA architecture, or more specifically, a target PLB architecture  $p$  and a Boolean function  $f$ , the Boolean matching problem either maps function  $f$  to PLB  $p$  by describing the appropriate configuration bits, or concludes that PLB  $p$  cannot implement function  $f$ . Most of the existing work for Boolean matching is based on function decomposition [6] or on canonicity and Boolean signatures [7]. These approaches are limited by the input size of the functions they can handle. Recently, a SAT-based approach [8] has been proposed to solve Boolean matching and was improved by [9] with a 3x speedup, and was further improved by [10] with up to 13x speedup. While SAT-based Boolean matching offers great flexibility in handling various FPGA architectures, it still suffers from long runtimes due to high computational complexity. For example, the Boolean matching procedure is called over 50,000 times for the MCNC circuit *i10* with less than 3000 gates, with a typical runtime for completing one SAT-based Boolean matching [8] for a 9-input sub-circuit at more than 20 seconds. It would appear that the runtime for heterogeneous FPGA technology mapping is prohibitively high due to the inefficiencies of Boolean matching.

Inspired by a recent improvement on Boolean matching for ASIC [11] and an enhancement on SAT reasoning [12] by exploring symmetries, we re-visit the SAT-based Boolean matching problem for heterogeneous FPGAs for orders of magnitude speedup over the existing algorithms [8, 9, 10]. We propose to leverage function and architecture symmetries explicitly during CNF (conjunctive normal form) encoding and dramatically reduce the SAT problem size and the SAT reasoning runtime. The experimental results show that the proposed algorithm obtains up to 200x speedup by considering symmetries compared to the original algorithm [8], while the recent papers [9, 10] obtained merely up to 13x speedup.

The rest of this paper is organized as follows: Section 2 formalizes the concepts involved in Boolean matching and reviews the SAT-based encoding [8]. Section 3 presents our heuristics for improving the efficiency of the SAT-based Boolean matching approach using symmetries. Section 4 details our experimental results, and section 5 concludes the paper. We include details of this paper in a technical report (<http://eda.ee.ucla.edu>).

## 2. BACKGROUNDS AND PRELIMINARIES

A *programmable logic block (PLB)*  $H(P)$  consists of a net-

\*This paper is partially supported by NSF grant CCR-0306682. Address comments to [lhe@ee.ucla.edu](mailto:lhe@ee.ucla.edu).

work of interconnected non-programmable and programmable logic devices with a set  $P$  of input pins  $\{p_1, \dots, p_m\}$ . Occasionally we may omit the set of input pins and simply use  $H$  to refer to the PLB  $H(P)$ . We consider two kinds of programmable logic devices in this paper: the  $K$ -input LUT and the  $K$ -input multiplexer (MUX). A  $K$ -LUT consists of  $K$  inputs, one output, and  $2^K$  configuration bits. A  $K$ -MUX consists of  $K$  inputs, one output, and  $\lceil \log K \rceil$  configuration bits.

The *Boolean matching* problem takes as input a PLB  $H(P)$  and a Boolean function  $f(X)$  over the variables  $X$  such that  $|X| \leq |P|$ , and determines whether the PLB  $H(P)$  can implement the function  $f(X)$ .

For the simple case where  $H$  is a  $K$ -LUT, any function  $f(X)$  where  $|X| \leq K$  can be implemented by the  $K$ -LUT. When  $H$  contains multiple LUTs however, the question becomes non-trivial.

## 2.1 SAT Encoding for Boolean Matching

[8] presented an algorithm to encode the Boolean matching problem for heterogeneous PLBs into a Boolean function in conjunctive normal form (CNF), which can be solved by SAT reasoning. As a review of the algorithm, we consider the example PLB in Figure 1(b), containing a LUT-2 and an AND-2 gate.

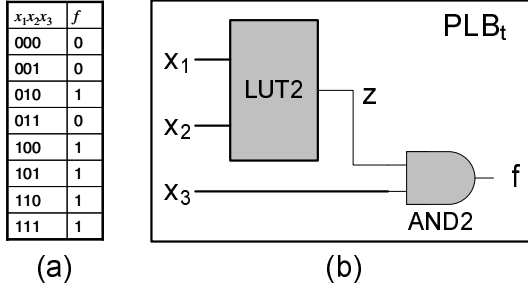


Figure 1: (a) Truth table for  $f$ , (b) Target PLB

To test whether function  $f$ , whose truth table is shown in Figure 1(a), can be implemented by this PLB, let  $X = \{x_1, x_2, x_3\}$  be the set of input pins. We generate a SAT instance in below steps:

1. Create CNF formulas for individual elements in  $PLB_t$ .

$$\begin{aligned}
 G_{LUT} &= (x_1 + x_2 + \neg L_1 + z)(x_1 + x_2 + L_1 + \neg z) \\
 &\quad (x_1 + \neg x_2 + \neg L_2 + z)(x_1 + \neg x_2 + L_2 + \neg z) \\
 &\quad (\neg x_1 + x_2 + \neg L_3 + z)(\neg x_1 + x_2 + L_3 + \neg z) \\
 &\quad (\neg x_1 + \neg x_2 + \neg L_4 + z)(\neg x_1 + \neg x_2 + L_4 + \neg z) \\
 G_{AND} &= (z + \neg o) \cdot (x_3 + \neg o) \cdot (\neg z + \neg x_3 + o)
 \end{aligned} \quad (1)$$

2. The characteristic function of the PLB is then formulated as:

$$G = G_{LUT} \cdot G_{AND} \quad (2)$$

Notice that the output variable is called  $o$ .

3. Replicate (2) to remove the universal quantifiers on the input variables in  $X$ . This formulates  $G_{SAT}$  as:

$$\begin{aligned}
 G_{SAT} &= G[X/000, o/0, z/z_1] \cdot G[X/001, o/0, z/z_2] \cdot \\
 &\quad G[X/010, o/1, z/z_3] \cdot G[X/011, o/0, z/z_4] \cdot \\
 &\quad G[X/100, o/1, z/z_5] \cdot G[X/101, o/1, z/z_6] \cdot \\
 &\quad G[X/110, o/1, z/z_7] \cdot G[X/111, o/1, z/z_8]
 \end{aligned} \quad (3)$$

A satisfiable assignment to  $G_{SAT}$  implies that  $f$  can be implemented by the PLB. In this particular case the SAT solver indicates

that this problem is unsatisfiable; therefore the Boolean function specified by the truth table in Figure 1(a) cannot be implemented by  $PLB_t$  in Figure 1(b).

## 2.2 Input Permutations

An important issue in Boolean matching is *input permutation*, which allows different mappings from pins in a PLB to variables of a Boolean function. For example, function  $f_a = x_1 \cdot (x_2 + x_3)$  and function  $f_b = x_3 \cdot (x_1 + x_2)$  are equivalent under input permutation, i.e., function  $f_a$  can be transferred to  $f_b$  by the input permutation  $\tau = (3, 2, 1)$ . However,  $f_a$  cannot be implemented by  $PLB_t$  in Figure 1(b) by mapping the input variables of  $f_a$  and the input pins of  $PLB_t$  in the same order, while  $f_b$  can.

In practice, input permutation must be considered in FPGA design during Boolean matching in order to maximize the number of implementable instances. However, the number of permutations for a  $K$ -input Boolean function is  $K!$  which grows extremely quickly. In order to consider input permutations in the SAT formulation, [8] proposed to add programmable MUXs before each primary input of the target PLB (see Figure 2). All possible permutations are encoded by these MUXs. For each of these programmable MUXs,  $\lceil \log n \rceil + 1$  additional variables are needed to represent the configuration bits (e.g.,  $L_{11}, L_{12}, L_{21}, L_{22}, L_{31}, L_{32}$  in Figure 2) and the intermediate pins (e.g.,  $z_1, z_2, z_3$ ), and  $O(n^2)$  additional clauses are needed as well. Thus, considering input permutations adds  $O(n^3)$  clauses and  $n \cdot (\lceil \log n \rceil + 1)$  variables to the original formulation. In practice, the size of a LUT is usually less than six, so adding these MUXs can double the size of the SAT problem if  $n$  is a large (i.e., greater than six) and the computational cost will be increased significantly.

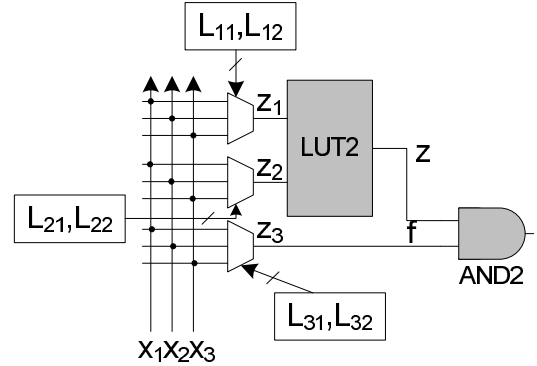


Figure 2: Considering input permutations with additional MUXs

## 3. SPEEDUP BY SYMMETRIES

By considering symmetries in the SAT encoding, the permutation MUXs can be eliminated and the computational cost can be reduced.

### 3.1 Symmetries in Boolean Functions

Variable  $x_i$  and  $x_j$  of Boolean function  $f(x_1, \dots, x_n)$  are *symmetric* if the truth table of  $f$  remains the same when  $x_i$  and  $x_j$  are swapped, i.e., if  $f(\dots, x_i, \dots, x_j, \dots) = f(\dots, x_j, \dots, x_i, \dots)$ . We can consider only *distinct permutations* by observing the variable symmetries exhibited in a Boolean function, making the programmable MUXs added in subsection 2.2 unnecessary.

Given an  $n$ -input Boolean function  $f(x_1, \dots, x_n)$ , we first test the symmetries of every input pair  $(x_i, x_j)$  by comparing the truth tables before and after swapping variables  $x_i$  and  $x_j$ . Using the

symmetric relationships between every variable pair, we can construct an undirected graph with nodes representing each variable, and an edge connecting each pair of symmetric variables. For example, consider a 9-input Boolean function having four symmetries (0, 1, 6, 8), (3, 4, 5), (2), and (7) as shown in Figure 3. For any two permutations  $\tau_1$  and  $\tau_2$ , if the only difference between them is within the same symmetry cluster ((0, 1, 6, 8) or (3, 4, 5), in this example), we have  $\tau_1 = \tau_2$  and only one of them needs to be tested during Boolean matching. Therefore the number of distinct permutations under such symmetry is  $9! / (4! \times 3! \times 1! \times 1!) = 2520$ , reducing the number of permutations to consider by a factor of 144.

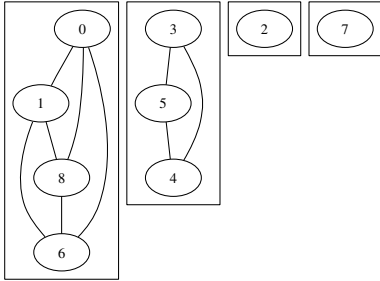


Figure 3: Symmetries in a 9-input Boolean function

Note that the time required to identify symmetries of an  $n$ -input function using the above algorithm is  $O(n^2 \cdot 2^n)$ . Although more sophisticated algorithms [13] can be used to detect symmetries, we find that the computational cost for symmetry detection is negligible compared to the Boolean matching time in our experiments. Taking advantage of the symmetries exhibited by a Boolean function allows us to significantly reduce the number of permutations to be tested.

### 3.2 Symmetries in PLB Architecture

Most commercial PLB architectures exhibit symmetries with respect to their input pins. Additional architectural symmetries may also exist and can be discovered by considering deeper logic levels of the circuit. Formally, we define *first order* and *second order architectural symmetries* as follows:

**DEFINITION 1. First Order Architectural Symmetry:** Any two input pins  $x_i, x_j$  connected directly to the same LUT are symmetric under the permutation  $(x_i, x_j)$ .

**DEFINITION 2. Second Order Architectural Symmetry:** The inputs  $x_1, \dots, x_k$  and inputs  $y_1, \dots, y_k$  for two  $k$ -input LUTs  $L_x$  and  $L_y$ , respectively, are symmetric under permutation  $\pi(y_{i_1}, \dots, y_{i_k}, x_{j_1}, \dots, x_{j_k})$  if the outputs  $x$  and  $y$  of these two LUTs are symmetric.

For example, in the PLB shown in Figure 4, the inputs  $x_1$  and  $x_2$  are symmetric, as are the inputs  $x_3$  and  $x_4$ , which means that ignoring the input permutations where they are swapped will not affect the decision of whether a certain Boolean function can be implemented by this PLB. The symmetries between  $x_1$  and  $x_2$  and between  $x_3$  and  $x_4$  are first order architectural symmetries. Furthermore, since the outputs of both LUTs feed into a 2-input AND gate whose inputs are symmetric, ignoring the configurations where two groups of pins  $(x_1, x_2)$  and  $(x_3, x_4)$  are swapped under the permutation  $\pi = (x_3, x_4, x_1, x_2)$ ,  $\pi = (x_3, x_4, x_2, x_1)$ ,  $\pi = (x_4, x_3, x_1, x_2)$  will not affect the Boolean matching decision. This is a second order architectural symmetry.

To detect symmetries exhibited in the PLB architecture, we extend the structural analysis algorithm presented in [14] to consider programmable logic devices in the circuit. Interested readers should refer to [15] for details. Architectural symmetry detection can be

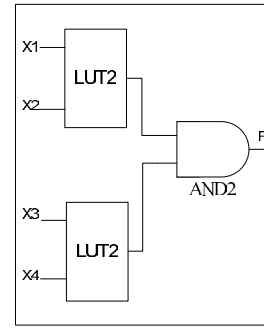


Figure 4: A second order symmetric PLB

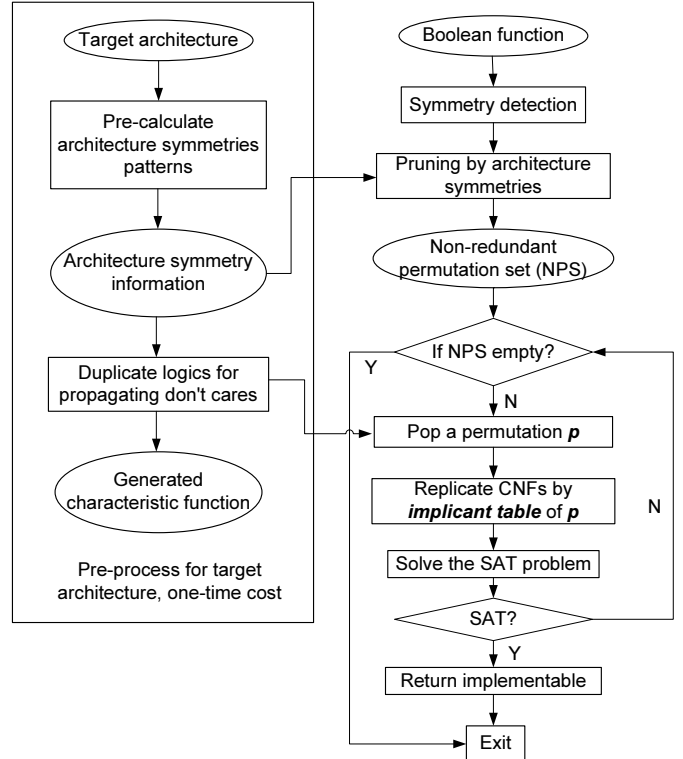


Figure 5: The flow of the overall algorithm

done in the pre-process before re-synthesis. Since PLB sizes are typically small, runtime cost is not an issue.

### 3.3 Overall Algorithm

Figure 5 shows the flow of our overall algorithm. We first pre-process the architecture of the target PLB by extracting its architectural symmetry information (using the algorithm in Section 3.2) and generate a template of the characteristic function for the PLB. For each Boolean function to be tested, we first detect the function symmetries (using the algorithm in Section 3.1) and prune the redundant permutations based on both architectural and function symmetries. Then each distinct permutation is tested individually by replication of the characteristic function. Given each permutation  $p$ , a SAT problem is generated by replicating the characteristic function based on the implicant table of  $p$  (the SAT encoding by implicant table was proposed in [10] and is summarized in the appendix). If any permutation gives rise to a satisfiable solution, then the Boolean function can be implemented by the target PLB. If instead none of the SAT instances are satisfiable, then the function cannot be implemented by the target PLB.

## 4. EXPERIMENTS

Our algorithm implementation is in C++ and Perl. The SAT solver used is miniSAT1.14 [16]. The implicant table-based SAT encoding [10] has been implemented and integrated into our algorithm as shown in Figure 5. To show the effectiveness of our improvement to the SAT-based Boolean matching algorithm, we extract over 10k fanout-free cones (FFCs) with 5-9 inputs from MCNC benchmarks based on the method presented in [3] as the Boolean functions. The target PLB architecture is similar to the PLB in Figure 4 except that the two input LUTs have four inputs and the output logic is a 3-input LUT instead of a 2-input AND gate. All experimental data are collected on a Linux server with a 1.9GHz Xeon CPU and 2GB memory.

For a nine-input PLB the total number of input permutations is  $9! = 362,880$ . In our experiments we observe that the number of distinct permutations to consider is typically reduced by over two orders of magnitude (100x) when employing Boolean function symmetries, and by another two orders of magnitude (100x) when considering architectural symmetries.

Table 1 compares the runtime of the original algorithm SAT-BM presented in [8] and our improved algorithm (SAT-IP). The runtime for producing SAT instances is not considered in the table. The average SAT instance sizes and runtimes of both algorithms are shown in the table. As the number of inputs in the Boolean function increases, the SAT instance size increases exponentially for SAT-BM. On the other hand, the size of each sub-SAT instance for our SAT-IP algorithm remains nearly the same regardless of the number of inputs in the Boolean function, and the number of unique permutations grows slowly. Compared to SAT-BM, SAT-IP achieves up to 400x overall speedup, where 200x speedup of SAT-IP is due to the consideration of symmetries and 2x more speedup is due to the integration of implicant table-based SAT-encoding. More significant speedup is expected if Boolean functions with wider inputs are considered. Note that two recent improvements on the SAT-based Boolean matching problem, [9] and [10] obtained up to 13x speedup compared to [8].

Testcases	func size	5.00	6.00	7.00	8.00	9.00
	problem#	1398	1981	2263	2172	2134
	variable#	867	1571	2979	5795	11427
SAT-BM	clause#	6945	13889	27777	55553	111105
[8]	runtime (s)	0.47	2.22	2.39	27.53	173.59
	variable#/inst	367	442	358	405	454
	clause#/inst	1509	1850	1466	1683	1906
SAT-IP	unique perm#	4.30	17.73	30.90	26.6	161.47
(ours)	runtime (s)	0.01	0.05	0.07	0.07	0.43
	speedup	45x	48x	32x	409x	403x

Table 1: SAT-BM vs. SAT-IP

To break down the effectiveness of each components in SAT-IP, we compare the runtimes of: the SAT-BM algorithm, the SAT-IP algorithm with truth table replication while considering only Boolean function symmetries, the SAT-IP algorithm with truth table replication while considering Boolean function symmetries and architectural symmetries and, the final SAT-IP algorithm with implicant table replication while considering all symmetries. The results (refer to [15] for detailed data) show that although Boolean function symmetry itself cannot bring significant speedup, combined with architectural symmetry optimizations, two orders of magnitude speedup can be attained compared to the original SAT formulation. With the integration of implicant table-based replication, an additional 2x speedup can be achieved. This is different from the 3-13x speedup reported in [10] and is due to the overlap in techniques for runtime reduction.

## 5. CONCLUSION

Targeting orders of magnitude speedup over the existing algorithms for SAT-based Boolean matching [8, 9, 10], we have presented an algorithm to significantly improve the efficiency of SAT-based Boolean matching by exploring the symmetries exhibited in both the Boolean function and the target PLB architecture. Considering function and architecture symmetries explicitly during CNF encoding, the SAT problem size and the SAT reasoning runtime are dramatically reduced. The experimental results show that the proposed algorithm obtains up to 200x speedup by considering symmetries compared to the original algorithm [8], while recent work [9, 10] obtained up to 13x speedup. Our future plans entail integrating our algorithm into technology mapping for FPGA architecture exploration.

## 6. REFERENCES

- [1] J. Cong and Y. Ding, "Flowmap: An optimal technology mapping algorithm for delay optimization in lookup-table based fpga designs," in *TCAD*, 1994.
- [2] "Abc: A system for sequential synthesis and verification," in <http://www.eecs.berkeley.edu/~alumni/abc/>.
- [3] J. Cong, C. Wu, and Y. Ding, "Cut ranking and pruning: Enabling a general and efficient fpga mapping solution," in *FPGA*, 1999.
- [4] "Xilinx product datasheets," in <http://www.xilinx.com/literature>.
- [5] D. Lewis and et al, "The stratix ii routing and logic architecture," in *FPGA*, Feb 2005.
- [6] J. Cong and Y.-Y. Hwang, "Boolean matching for lut-based logic blocks with applications to architecture evaluation and technology mapping," 2001.
- [7] L. Benini and G. D. Micheli, "A survey of Boolean matching techniques for library binding," *TODAES*, vol. 2, no. 3, pp. 193–226, 1997.
- [8] A. Ling, D. Singh, and S. Brown, "FPGA technology mapping: a study of optimality," in *DAC*, 2005.
- [9] S. Safarpour, A. Veneris, G. Baeckler, and R. Yuan, "Efficient sat based boolean matching for fpga technology mapping," in *DAC*, 2006.
- [10] J. Cong and K. Minkovich, "Improved sat-based boolean matching using implicants for lut-based fpgas," in *FPGA*, 2007.
- [11] Z. Wei, D. Chai, A. Kuehlmann, and A. R. Newton, "Fast boolean matching with dont cares," in *ISQED*, 2006.
- [12] F. Aloul, A. Ramani, I. Markov, and K. Sakallah, "Solving difficult instances of boolean satisfiability in the presence of symmetry," in *TCAD*, 2003.
- [13] J. S. Zhang, M. Chrzanowska-Jeske, A. Mishchenko, and J. R. Burch, "Generalized symmetries in boolean functions: Fast computation and application to boolean matching," in *IWLS*, 2004.
- [14] J. S. Zhang, A. Mishchenko, R. Brayton, and M. Chrzanowska-Jeske, "Symmetry detection for large boolean functions using circuit representation, simulation and satisfiability," in *DAC*, 2006.
- [15] Y. Hu, V. Shih, R. Majumdar, and L. He, "Exploiting Symmetry in SATBased Boolean Matching for Heterogeneous FPGA Technology Mapping," in *Technical Report UCLA Engr 07-265*, 2007.
- [16] N. Een and N. Sorensson, <http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/MiniSat.html>.
- [17] E. M. Sentovich et. al., "SIS: A system for sequential circuit synthesis," in *Department of Electrical Engineering and Computer Science, Berkeley, CA 94720*, 1992.