Efficient SAT-based Boolean Matching for FPGA Technology Mapping

Sean Safarpour, Andreas Veneris
Dept. of Electrical and Computer Engineering
University of Toronto
Toronto, ON, Canada
{sean, veneris}@eecg.toronto.edu

Gregg Baeckler, Richard Yuan
Altera Corporation
San Jose, CA, USA
{gbaeckle, ryuan}@altera.com

ABSTRACT

Most FPGA technology mapping approaches either target Lookup Tables (LUTs) or relatively simple Programmable Logic Blocks (PLBs). Considering networks of PLBs during technology mapping has the potential of providing unique optimizations unavailable through other techniques. This paper proposes a Boolean matching approach for FPGA technology mapping targeting networks of PLBs. To overcome the demanding memory requirements of previous approaches, the Boolean matching problem is formulated as a Boolean Satisfiability (SAT) problem. Since the SAT formulation provides a trade-off between space and time, the primary objective is to increase the efficiency of the SAT-based approach. To do this, the original SAT problem is decomposed into two easier SAT problems. To reduce the problem search space, a theorem is introduced to allow conflict clauses to be shared across problems and extra constraints are generated. Experiments demonstrate a 340% run time improvement and 27% more success in mapping than previous SAT-based approaches.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design Aids—*Automatic synthesis, Optimization*

General Terms

Algorithm, Design, Performance

Keywords

FPGA technology mapping, Boolean matching, Boolean satisfiability

1. INTRODUCTION

Field Programmable Gate Arrays (FPGAs) are integrated devices composed of a matrix of Programmable Logic Blocks (PLBs) and programmable routing resources. Unlike custom integrated circuits, the functionality of FPGAs can be defined after fabrication. The flexibility of FPGAs is due to the reconfigurability of the functionality and the connectivity of the PLBs. Modern PLBs usually include a network of simple gates, multiplexers (MUX) and k-input Lookup Tables (k-LUTs). A k-LUT is a logic block that can be programmed to implement any function of k-input variables.

DAC 2006, July 24–28, 2006, San Francisco, California, USA.

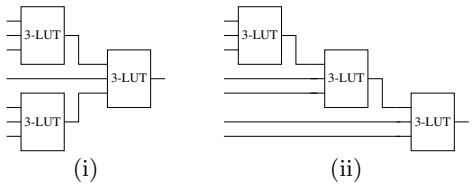


Figure 1: Two different mapping options

A crucial step in the overall FPGA Computer Aided Design (CAD) flow is *technology mapping* [3]. 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 representations. Depending on the technology mapping approach, the resulting network can exhibit different area, delay, and power costs. Today, much interest is placed in developing technology mapping techniques that minimize combinations of these optimization factors [3, 6, 12, 14].

One approach to technology mapping is known as *Boolean matching* [3, 6, 1]. Informally, Boolean matching for FPGAs can determine whether a function f can be implemented by a LUT or a PLB. Boolean matching can be employed as an integral part of technology mapping [7], during the resynthesis effort [14, 9], or as a means to evaluate different FPGA architectures [6, 9].

For the most part, previous FPGA technology mapping and Boolean matching work is concerned with mapping a function f into individual LUTs or "simple" PLBs [3, 6, 12, 14, 5]. Many of today's commercial FPGAs contain complex PLBs composed of a network of interconnected LUTs, MUX, and simple gates [2, 15]. To effectively use the available resources and improve performance, efficient technology mapping tools targeting complex PLBs or networks of PLBs are needed. For instance, recognizing that a function f can be mapped into the network of PLBs shown in Figure 1 (i) may result in a smaller delay than the network of PLBs in Figure 1 (ii).

Current Boolean matching techniques are very effective when targeting simple PLBs or LUTs but they may be impractical when targeting networks of PLBs. The primary reason is that popular Boolean matching techniques based on functional decomposition [6, 12, 14] or on canonicity and Boolean signatures [3, 1] are limited in the input size of the functions they can handle. For instance, most functional decomposition techniques using BDDs target functions with five to eight inputs [6, 14]. Much larger input functions required for networks of PLBs may demand a prohibitive amount of memory for building the BDDs. Recently, an approach was proposed formulating Boolean matching as a Boolean satisfiability (SAT) problem [9]. The SAT formulation may alleviate the memory explosion problem at the cost of long run times associated with searching the solution space.

In this paper, we propose a novel SAT-based Boolean matching approach for FPGA technology mapping targeted for networks of interconnected PLBs. Since networks of PLBs may have more inputs than a single PLB, the approach is viable for functions with greater than ten inputs. The benefit of targeting larger input functions and PLB networks is that unique optimization opportunities unexplored by previous techniques may arise. In this paper we are only concerned with the Boolean matching aspect of technology mapping. We further assume that any area, delay, and power evaluation techniques can be used to determine the best mapping for the given specifications [6, 12].

To effectively map larger input functions into a network of PLBs, we develop theory and heuristics that enhance and improve the performance of the SAT-based Boolean matching procedure from [9]. The major contributions of this paper are as follows.

- We develop a SAT-based two stage iterative approach where we perform a *coarse pin assignment* followed by a *detailed pin assignment*. An approximate mapping is quickly found during the coarse pin assignment while the detailed pin assignment refines or rejects this approximate solution. In this manner, we decompose the original computationally "hard" problem into two relatively "easier" subproblems. Furthermore, if the original problem does not have a solution for a given PLB network the method quickly aborts instead of exhaustively exploring the non-solution space. Intuitively, this process is analogous to the global and detailed routing techniques [4].
- We prove a theorem that allows the reuse of the highly valuable conflict clauses from different Boolean matching problem instances for a given set of PLBs. By collecting and reusing this information, future problems may benefit and be solved more efficiently.
- We enrich the original SAT formulation with additional constraint clauses to reduce the SAT solver's search space. These additional clauses restrict the SAT solver from exploring regions of the non-solution space.

Extensive experiments demonstrate that these contributions result in a run time improvement of 340% while finding 27% more solutions over previous approaches. The reduction in run time makes the proposed approach viable for re-synthesis of critical regions and FPGA architecture evaluation [6, 9]. The results also encourage further effort in SAT-based FPGA Boolean matching techniques.

This paper is organized as follows. The next section provides background information used throughout this paper. Section 3 presents the proposed SAT-based Boolean matching approach while Section 4 develops search space reduction techniques. Section 5 demonstrates the benefits of the contributions through extensive experiments and Section 6 concludes this work.

2. PRELIMINARIES

2.1 Boolean Matching

In this paper we refer to a network of interconnected PLBs as a *configuration*. For instance, Figure 1 (i) and (ii) depicts two different configurations. A configuration $H(P)$ has m input pins where $P = \{p_1, p_2, ..., p_m\}$. To distinguish between the n inputs of function $f(X)$, $X = \{x_1, x_2, ..., x_n\}$, and the configuration input pins, we refer to the latter simply as *pins*. We also use H to refer to $H(P)$. Using this terminology, the problem of interest can be now defined as follows:

Problem Definition: Boolean matching is the process of determining whether a configuration $H(P)$ can implement a function $f(X)$ for $|X| = n \leq m = |P|$.

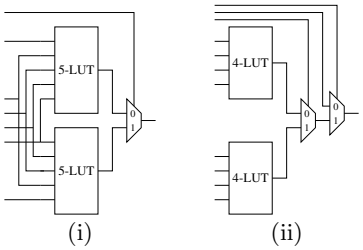


Figure 2: Industrial FPGA PLBs

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. For PLBs such as those shown in Figure 2 (i) and (ii), for example, it is not obvious what 7-input functions and 11-input functions can be implemented in each configuration, respectively. As discussed earlier, it may be beneficial from an area and delay point of view to implement all 7 and 11-input functions via the configurations in Figure 2 (i) and (ii) whenever possible.

2.2 Boolean Satisfiability

Boolean satisfiability is the problem of determining whether there exists a variable assignment to a Boolean formula Φ such that it evaluates to `true`. Formula Φ contains a set of variables and connectives such as ¬(not), $\cdot$ (and), $+$ (or), $\rightarrow$ (implies), etc. If such a variable assignment exits, Φ is said to be satisfiable or `SAT`, otherwise it is unsatisfiable or `UNSAT`. For most modern SAT solvers the problem formula Φ is provided in *Conjunctive Normal Form* (CNF), that is, a conjunction of *clauses* where each clause is a disjunction of *literals*. A literal is an instance of a variable or its negation. The size of a clause c, denoted by $|c|$ is equal to the number of literals in the clause. Since a formula is `SAT` if all its clauses evaluate to `true`, at least one literal in each clause must evaluate to `true`. The following is an example of a CNF formula Φ and a satisfying variable assignment.

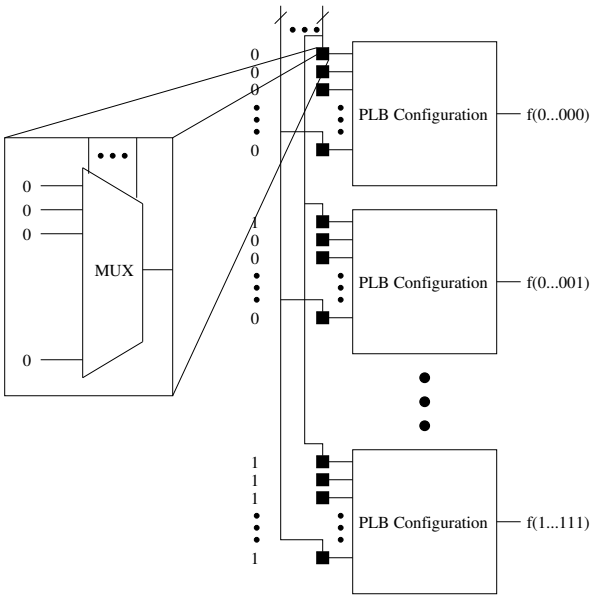
$$\Phi = (a + \overline{b} + c) \cdot (b) \cdot (\overline{a} + \overline{c}) \cdot (a + \overline{d})$$
$$a = 1, b = 1, c = 0, d = 1$$

There exist fast procedures to convert circuits composed of primitive gates to CNF in linear time [8]. For this reason, we do not distinguish between a circuit and its corresponding SAT formulation as the latter can be obtained from the former. Modern SAT solvers implement well-studied algorithms with optimized branch-and-bound procedures such as intelligent decision making, conflict based learning, and non-chronological backtracking to solve large SAT problems with hundreds of thousands of clauses and variables in an efficient manner [11].

During the SAT solving procedure, most SAT solvers "learn" from their mistakes when exploring part of the non-solution space by generating *conflict clauses*. Once a conflict is discovered by the SAT solver, a conflict clause is generated and added to Φ to prevent re-exploring this part of the non-solution space. As an example, consider a situation where the SAT solver has made the unsatisfiable variable assignments $\{a = 0, b = 1, c = 1\}$. As a result of this assignment, the conflict clause $(a + \overline{b} + \overline{c})$ is added to the CNF Φ to prevent this combination from re-occurring in the future.

2.3 Basic SAT-based Boolean Matching

In this section we describe the basic formulation of the Boolean matching problem as a SAT problem and refer to it as the *standard* approach. The standard approach presented here is a slight variation of the one proposed in [9]. To determine whether an n input function can be implemented in a configuration with m pins, we must consider all possible function input to pin mappings. For the simple case where $n = m$, there are $n!$ possible mappings since each function input can be mapped to each configuration pin.



$$\Phi = \prod_{d=0}^{2^{|X|}-1} CNF(H_d) \cdot CNF(MUX(X)_d) \cdot BIN(d) \cdot f(d)$$

Figure 3: Standard Boolean matching formulation

Formulating this problem in CNF requires $O(n!)$ clauses. For example, an 11-input function generates more than 39 million clauses. To avert this large memory requirement, a hardware block, $MUX(X)$, is used to represent the $n!$ mappings. This hardware consists of a multiplexer for every configuration pin, where the inputs of the multiplexer are the function inputs and the output of the multiplexer is a configuration pin. The select line of the multiplexers "represent" the mapping of the function inputs onto the configuration pins. The CNF of the target hardware configuration H with the multiplexers is given by $CNF(H) \cdot CNF(MUX(X))$.

The next step consists of constraining the hardware configuration to implement the desired behavior of function $f(X)$ for all possible values of the input X. For example, if $f(X)$ is a function of 2 variables, then the hardware configuration must implement the correct output for all input combinations $X = 00$, $X = 01$, $X = 10$, $X = 11$.

These constraints are formulated in CNF by replicating $2^{|X|}$ times the formula $CNF(H) \cdot CNF(MUX(X))$. We refer to each replication as a *stamp*. For each stamp d, where $d = 0...2^{|X|}-1$, the input of the $MUX(X)_d$ multiplexers are constrained to the binary value corresponding to d denoted as $BIN(d)$ and the output of the H_d is constrained to binary value $f(d)$. Note that for each configuration pin, the select lines of each $MUX(X)$ are tied together for all stamps. This allows for a consistent mapping across the stamps. Figure 3 represents the different blocks of the standard formulation and the entire CNF formula Φ.

If the CNF formula Φ is found to be `UNSAT` by a SAT solver, it means that the hardware configuration H cannot implement the function f. If the problem is `SAT`, then the SAT solver also returns a satisfying assignment to all the formula variables thus providing a solution to the Boolean matching problem (*i.e.,* logic values for the select line of $MUX(X)$ and the k-LUT configuration bits, if H contains k-LUTs).

3. PROPOSED SAT-BASED BOOLEAN MATCHING APPROACH

In general, the standard formulation may suffer from long execution times. The main reason for the long run times is that the SAT solver must search through all the possible

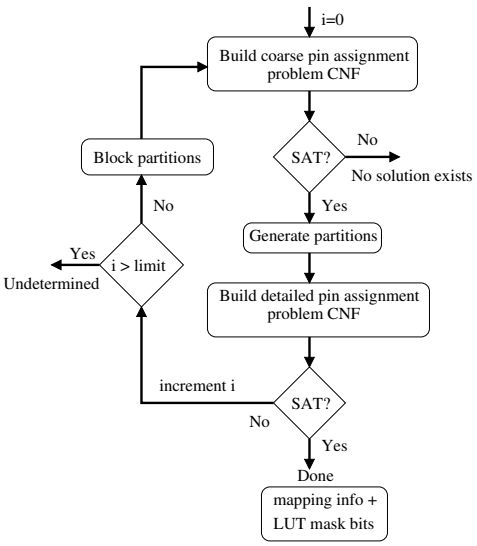


Figure 4: Proposed Boolean matching approach

mapping combinations and permutations to find a satisfying assignment to the LUT configuration bits. In this section we reduce the complexity of the standard formulation by proposing a novel SAT-based Boolean matching approach.

Consider the case where a k-input function is mapped to a simple k-LUT. In this case, the exact mapping of the function inputs to configuration pins is not necessary since a valid set of LUT configuration bits can be found for all permutations of the function inputs to the k-LUT pins. For instance, a function $f(x_1, x_2) = \overline{x_1} \cdot x_2$ can be mapped to a 2-LUT with configuration bits $\{0, 1, 0, 0\}$ where x_1 and x_2 are assigned to pins 1 and 2 of the LUT. Similarly a second mapping requires configuration bits $\{0, 0, 1, 0\}$ where x_2 and x_1 are assigned to the LUT pins 1 and 2, respectively.

As demonstrated by the above example, for configurations composed of only k-LUTs the problem consists of finding which function inputs are mapped to which k-LUTs (if many k-LUTs exist). In other words, *partitioning* the function inputs into groups corresponding to the k-LUTs (without determining the individual mapping onto the k-LUT pins) guarantees the existence of a solution. Based on the above observation, we propose a new SAT-based Boolean matching approach which breaks the problem in two stages: a *coarse pin assignment* stage and a *detailed pin assignment* stage.

In the coarse pin assignment stage, the targeted configuration H is simplified by replacing all k-input PLBs with k-LUTs. That is, we compute an approximation by assuming that every k-input PLB *can* implement any function of k inputs. These k-LUTs are used to create the $2^{|X|}$ stamps of the standard approach as described in Section 2.3. The resulting CNF is essentially a much easier problem which is independent of any mapping permutations. The solution of the coarse pin assignment determines how the function inputs may be partitioned among the different PLBs within a given configuration.

In the detailed pin assignment stage, a fine grain mapping is performed for the given sets of partitions in the first stage. The partition restrictions are encoded in CNF and added to the standard formulation. The added restrictions ensure that each function input is mapped to a PLB pin in the assigned partition. As a result of the partition restrictions, much fewer permutations are allowed in the second stage than that of the standard formulation. For instance, partitioning 11 input variables into two equal sets leads to $O(|X/2|!)$=720 permutations for each problem which is orders of magnitude less than $O(|X|!)$=39,000,000 in the standard approach.

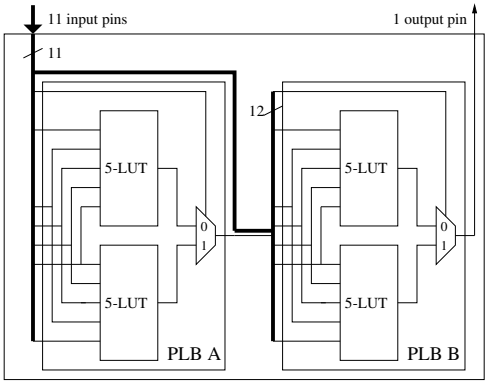


Figure 5: Configuration with two Stratix II PLBs

The above approach has an additional advantage. Since the coarse pin assignment stage performs an over-approximation, an `UNSAT` outcome in the first stage signifies that function f cannot be implemented in configuration H. A fast `UNSAT` response is very beneficial in practice because functions that *do not* have a feasible mapping to a given configuration are ruled out relatively quickly thus avoiding unnecessary computations. If the outcome is `SAT`, then the approach proceeds to the detailed pin assignment stage. An `UNSAT` result in the second stage indicates that there does not exist a mapping given the current partitions and another partitioning is sought by iterating back to the coarse pin assignment stage. On the other hand, if the result of the second stage is `SAT`, a valid mapping and corresponding LUT configuration bits are returned by the SAT solver thus completing the process. Note that due to performance concerns it may be desirable to terminate this process after a limited number of iterations. Figure 4 illustrates the proposed framework described above.

3.1 Example

The following example illustrates the coarse and detailed pin assignment stages of the Boolean matching approach.

Consider the 11-input configuration H based on two Altera Stratix II PLBs shown in Figure 5. Notice that the inputs to PLB B can be any of the 11 inputs as well as the output pin of PLB A. This configuration has been found to provide area, delay or power benefits for many common 11-input functions [2]. However, not all 11-input functions can be implemented using this configuration.

The first step of the proposed approach is to convert the 5-LUT and MUX combinations of each 7 input PLB into a 7-LUT. The resulting configuration, shown in Figure 6, is used to generate the 2^{11} stamps described in Section 2.3 and the corresponding CNF formula Φ.

Assuming that the result of the first stage is `SAT`, the SAT solver returns a partitioning for each 7-LUT. The first partition corresponds to the function inputs assigned to PLB A, and the second partition corresponds to the function inputs assigned to PLB B. Notice that partition B only contains 6 inputs because the output of PLB A is always an input to PLB B. The partitions are shown below.

$PartitionA = \{x_1, x_6, x_9, x_2, x_4, x_3, x_8\}$
$PartitionB = \{x_3, x_5, x_{10}, x_9, x_7, x_{11}\}$

In the detailed pin assignment stage, constraints are generated to prevent the function inputs assigned to $PartitionA$ ($PartitionB$) to map to PLB B (PLB A). The CNF of the standard formulation is enriched with the partition constraints and solved by the SAT solver. In this stage, the SAT solver finds a "detailed" mapping of the function inputs in each partition to the pins of the corresponding PLB. For this example, the SAT solver returns `SAT` leading to an assignment to the LUT configuration bits and to the following mapping.

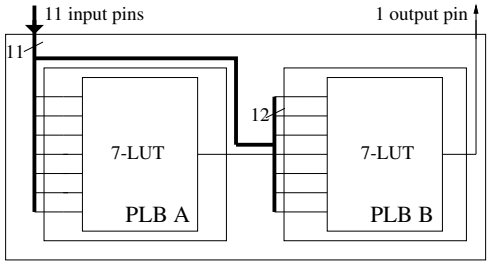


Figure 6: Transformation of two Stratix II PLBs in the first stage

Mapping for $PartitionA$		Mapping for $PartitionB$	
x_2 :	PLB A pin 1	x_3 :	PLB B pin 1
x_9 :	PLB A pin 2	x_{10} :	PLB B pin 2
x_8 :	PLB A pin 3	x_5 :	PLB B pin 3
x_6 :	PLB A pin 4	PLB A output :	PLB B pin 4
x_4 :	PLB A pin 5	x_7 :	PLB B pin 5
x_1 :	PLB A pin 6	x_{11} :	PLB B pin 6
x_3 :	PLB A pin 7	x_9 :	PLB B pin 7

4. SEARCH SPACE REDUCING

The two stage coarse and detailed pin assignment approach can increase the efficiency of the overall Boolean matching problem. Performance can be further improved by reducing the SAT solver's search space. In this section we present two methods of increasing its efficiency based on enriching the problem CNF Φ with additional constraining clauses.

4.1 Reusing Conflict Clauses

During the search procedure, a SAT solver may run into conflicts due to unsatisfiable variable assignments. Modern SAT solvers learn from these conflicts by generating conflict clauses that prevent them from re-exploring these non-solution regions. It is well accepted that conflict clauses guide SAT solvers efficiently towards their final `SAT` or `UNSAT` outcome [11]. Conflict clauses are usually specific to the instance of the problem solved and cannot be always shared between different problems.

In this section, we show that the Boolean matching SAT formulation allows us to reuse conflict clauses across problems for a given configuration H. The following theorem formally states that it is safe to share conflict clauses across different problems as long as only the clauses representing the output value of the function f are different (the $f(d)$ components of Φ).

Theorem: Consider two CNF formulae Φ_1 and Φ_2 such that $\forall$ clause $c \in \{\Phi_1 - \Phi_2, \Phi_2 - \Phi_1\}$, $|c| = 1$ and $\exists c_i = \overline{c}$ where $c \in \Phi_1$ and $c_i \in \Phi_2$ or $c \in \Phi_2$ and $c_i \in \Phi_1$. Further assume that CNF Φ_c represents the conflict clauses deduced from Φ_1 [10]. The CNF $\{\Phi_2 \cup \Phi_c\}$ is satisfiable if and only if the CNF Φ_2 is satisfiable.

Proof: This theorem is similar to the claim in [13] stating that conflict clauses deduced from $\{\Phi_1 \cap \Phi_2\}$ may be used in either Φ_1 or Φ_2 without modifying the satisfiability outcome. The proof in [13] is based on the fact that all literals in the conflict clauses of Φ_c are deduced from common clauses to both problems, Φ_1 and Φ_2. In our case, we want to consider the above case plus the condition where literals many not be deduced from common clauses as long as they are deduced from clauses with size 1 (single literal).

The cases that we need to prove are (i) if $\{\Phi_2 \cup \Phi_c\}$ is `UNSAT` then Φ_2 is `UNSAT` and (ii) if $\{\Phi_2 \cup \Phi_c\}$ is `SAT` then Φ_2 is `SAT`. For both these cases, it is sufficient to show that Φ_c evaluates to `true` in the CNF $\{\Phi_2 \cup \Phi_c\}$. To proceed, we show that any conflict clause $c_c \in \Phi_c$ that contains a variable v which is also a clause c_v such that $|c_v| = 1$ and $c_v \in \Phi_1$ is satisfiable in Φ_2. Assuming without loss of generality that the literal for variable v appearing in the conflict clause is l, then the literal $\overline{l}$ must appear in $c_v \in \Phi_1$ [10].

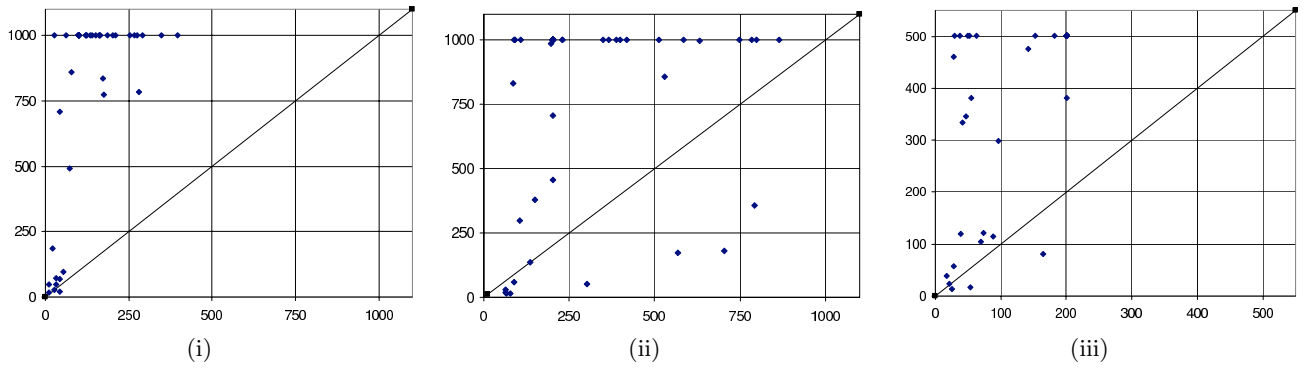


Figure 7: Run time comparison (s) of proposed approach (x-axis) with standard approach (y-axis)

Since the clause $c_v \in \{\Phi_1 - \Phi_2\}$, then there exists a clause $c_w \in \{\Phi_2 - \Phi_1\}$ such that $c_w = \overline{c_v}$. As a result, the literal $\overline{\overline{l}} = l$ must be satisfied in Φ_2 which will also satisfy the conflict clause Φ_c (since l appears in the conflict clause). ■

The above theorem allows us to generate a database of conflict clauses using k-input functions for each configuration of interest and reuse these clauses when matching any k-input functions on the given configurations.

4.2 Constraining the Search Space

Given the SAT formulation for Boolean matching described in Section 3, extra constraints can be developed to reduce the search space explored by the SAT solver. These constraints encoded in CNF reduce overall complexity of the problem by limiting the assignment possibilities of the MUX select lines.

First, we develop a set of constraints to ensure that each function input is mapped to a configuration pin. In other words, a function of n inputs mapped to a configuration with $m \geq n$ pins must indeed uses all n inputs. The CNF formulation demands that each of the n function inputs be selected by at least one of the m configuration pins. The overall space complexity of these constraints is thus $O(n \times m)$ clauses. For values of n and $m < 100$, the benefits associated with adding the extra clauses outweigh the overhead introduced as shown in the experiments.

Next, we develop an additional set of constraints concerned with distributing the function inputs across the available PLBs. For instance, assigning the same function input to two different pins on the same k-LUT does not use the full capacity of the k-LUT. As a result, we generate CNF formulae to ensure that no two configuration pins on the same LUT use the same function input. For a k-LUT, this leads to an $O(k^2)$ number of additional clauses. Similar to the first set of constraints, for relatively small values of $m < 100$, the benefits of reducing the search space are much greater than the costs associated to the extra clauses.

Note that both these sets of constraints are applied to the coarse and detailed pin assignment stages of the Boolean matching approach.

5. EXPERIMENTS

In this section we present a summary of the experimental results obtained for the SAT-based Boolean matching approach presented here. We evaluate the run time performance of the approach on 50 11-input functions extracted from the Altera customer design database. The experiments are conducted on a 3GHz Xeon processor with 1GB of memory and the SAT solver used is zChaff version 2004.5.13[11]. Each function is mapped to the three hardware configurations (a), (b) and (c) shown in Figure 5 and Figure 9 (i) and (ii), respectively. These configurations are based on the Altera Stratix II FPGA PLBs [2]. Configuration (a) is of interest because it provides a feasible implementation of many common 11-input functions. Configuration (b) may implement more functions than configuration (a) due to the extra 4-LUT. Configuration (c) is also desirable as it can provide delay or power benefits for many functions.

To demonstrate the benefit of the proposed approach against the standard approach, we execute both tools on all the functions using the three configurations. The results are plotted in Figure 7 (i), (ii), and (iii) for configurations (a), (b), and (c), respectively. Each point in the figures represents the run time in seconds taken by the standard approach (the y-coordinate) and by the proposed approach (the x-coordinate). Points lying above the diagonal line represent functions for which our proposed approach outperforms the standard approach. Notice that points aligned across the top of the graphs represent cases where the standard approach times out. Due to the over-approximation of the proposed approach, `UNSAT` problems which correspond to functions that cannot be mapped to a given configuration can be quickly identified without resulting in a timeout. Since approximately 90% of points in Figure 7 are above the diagonal lines, it is clear that proposed Boolean matching approach results in better run time performance than the standard approach.

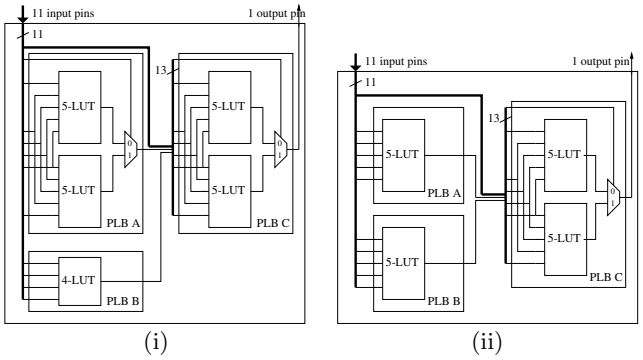


Figure 9: Configurations (b) and (c)

Next, we evaluate the benefit of the search space pruning techniques discussed in Section 4. Here extra clauses are generated and added to the problem CNFs. Conflict clauses are collected from executing the proposed approach on ten random functions with a timeout limit of 200 seconds. Of all the conflict clauses deduced, only those with fewer than 5 literals are recorded and added to the problem CNF. Constraint clauses used to reduce the problem search space are generated one time and reused for all problems. Note the time required to generate these clauses is negligible. The resulting performance with and without the search space reduction techniques is presented in Figure 8.

Figure 8 (i), (ii), and (iii) represent the results for configurations (a), (b), and (c), respectively. There are two curves plotted on each graph; the higher curve represents the tool with the search space reduction techniques while the lower curves represents the tool without these techniques. These curves demonstrate what fraction of functions can be mapped to the corresponding configuration under a given

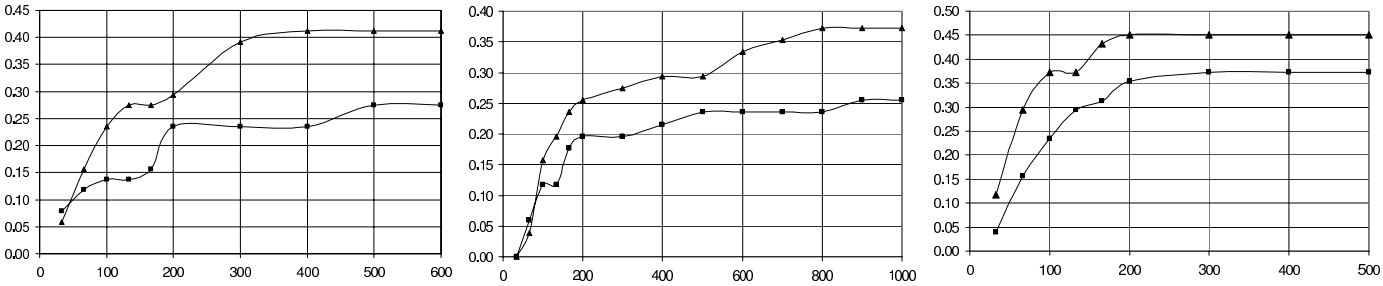


Figure 8: Impact of search space pruning techniques: percentage of functions mapped (%) vs. run time (s)

conf.	Max # sols	standard		basic proposed					complete proposed				
		# sol	CPU (s)	# itr	# sol	CPU (s)	% sol	% time	# itr	# sol	CPU (s)	% sol	% time
(a)	26	15	805	4.02	14	182	-7	442	3.75	21	166	40	485
(b)	30	18	776	3.41	13	231	-38	336	3.24	19	301	6	258
(c)	26	17	400	1.65	19	175	12	227	1.47	23	141	35	284
avg.	27.3	16.7	660.3	3.03	15.3	196.0	-11	335	2.80	21.0	202.7	27	342.3

Table 1: Results of proposed Boolean matching approach

time limit. For example, the point (200, 0.30) on the top curve of Figure 8 (i), states that running the proposed approach for 200 seconds results in 30% of the functions being mapped. Similarly, allowing the tool to run for an extra 100 seconds will result in approximately 40% of the functions being mapped. Generally, higher curves and steeper slopes before reaching a plateau indicate a more efficient tool since it can map more functions in shorter time. As illustrated in Figure 8, at approximately 100 seconds, the search space reduction techniques result in about 60% more mappings.

Table 1 summarizes the performance results of the standard method, the basic proposed approach, and the one with the search space reduction techniques on the three configurations and the 50 11-input functions. Each row of the table corresponds to a configuration stated in column one. Column two displays the maximum number of functions that can be possibly mapped to the configurations. The columns labeled "# sol" and "CPU (s)" present the number of mapped solutions found and the average time required to find the solutions in seconds for each of the three approaches. For the proposed basic and the complete approaches, the columns labeled "# itr" state the average number of iterations required by each approach. This value corresponds to the number of times the coarse and detailed pin assignment stages are performed. The columns labeled "% time" and "% sol" indicate the percentage improvement for the run time and the improvement in number of solutions of the proposed technique over the standard approach.

The over-approximation technique used in the first stage is crucial for the overall performance of the proposed approach. If the approximation is too loose, then time is wasted by performing many iterations of the approach only to reject the unsuitable solutions in the second stage. On the other hand, if the approximation is too tight, then problem is similar to solving the standard problem which may be computationally expensive. The relatively small number of iterations performed along with the reduction in run times demonstrates that the approximation technique developed is quite effective.

The final row of Table 1 presents the average numbers of the above data. Comparing the average values of "% time" and "% sol" for the complete proposed approach and the standard approach, we notice an approximate run time improvement of 340% while providing 27% more mappings.

6. CONCLUSION

We develop a two staged SAT-based formulation for the FPGA Boolean matching problem for networks of PLBs. In the first stage the function inputs are partitioned into groups corresponding to PLBs while in the second stage the function inputs are mapped to PLB pins. We further develop a theorem which allows us to reuse beneficial conflict clauses from different problems and reduce the problem search space by enriching the problem CNF with additional constraints. Experiments demonstrate a 340% run time improvement and 27% more mappings over previous methods.

7. ACKNOWLEDGMENT

The authors would like to thank Dr. Jason Anderson and Dr. Babette van Antwerpen for their insightful comments and technical suggestions which helped strengthen this work.

8. REFERENCES

[1] A. Abdollahi and M. Pedram. A new canonical form for fast Boolean matching in logic synthesis and verification. In *Design Automation Conf.*, pages 379–384, 2005.

[2] Altera Corp. Statix II device family data sheet. http://www.altera.com/literature/hb/stx2/stx2_sii5v1_01.pdf, 2005.

[3] L. Benini and G. D. Micheli. A survey of Boolean matching techniques for library binding. *ACM Trans. Des. Autom. Electron. Syst.*, 2(3):193–226, 1997.

[4] S. Brown, J. Rose, and Z. Vranesic. A detailed router for field-programmable gate arrays. *IEEE Trans. on CAD*, 11(5):620–628, 1992.

[5] J. Cong and Y. Ding. An optimal technology mapping for delay optimization in lookup-table based fpga designs. In *Int'l Conf. on CAD*, pages 48–53, 1992.

[6] J. Cong and Y. Y. Hwang. Boolean matching for LUT-based logic blocks with applications to architecture evaluation and technology mapping. *IEEE Trans. on CAD*, 20(9):1077–1090, 2001.

[7] K. Keutzer. DAGON: Technology binding and local optimization by DAG matching. In *Design Automation Conf.*, pages 341–347, 1987.

[8] T. Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Trans. on CAD*, 11:4–15, 1992.

[9] A. Ling, D. Singh, and S. Brown. FPGA technology mapping: a study of optimality. In *Design Automation Conf.*, pages 427–432, 2005.

[10] J. Marques-Silva and K. Sakallah. GRASP – a new search algorithm for satisfiability. In *Int'l Conf. on CAD*, pages 220–227, 1996.

[11] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conf.*, pages 530–535, 2001.

[12] C. Scholl. *Functional Decomposition with Application to FPGA Synthesis*. Kluwer Academic Publishers, 2001.

[13] O. Strichman. Pruning techniques for the sat-based bounded model checking problem. In *CHARME*, pages 58–70, 2001.

[14] N. Vemuri, P. Kalla, and R. Tessier. BDD-based logic synthesis for LUT-based fpgas. *ACM Trans. on Design Automation of Electronic Systems*, pages 501–525, 2002.

[15] Xilinx. Virtex-4 user guide. http://direct.xilinx.com/bvdocs/userguides/ug070.pdf, 2005.