FPGA Architecture Evaluation and Technology Mapping using Boolean Satisfiability

Andrew Ling
Department of Electrical and Computer Engineering
University of Toronto
Toronto, CANADA
aling@eecg.toronto.edu

Deshanand P. Singh,
Valavan Manohararajah,
Stephen D. Brown
Altera Corporation Toronto Technology Centre
Toronto, CANADA
dsingh|vmanohar|sbrown@altera.com

ABSTRACT

This paper presents a technology mapping algorithm that can be used to evaluate the robustness of any FPGA programmable logic block (PLB). This algorithm, named **SATMAP**, uses Boolean satisfiability (SAT) to determine if a logic cone can be implemented in a given PLB. This algorithm is a fundamental tool needed to study the utility of any proposed FPGA logic block. Our approach is the first tool of its kind that allows radical new features of FPGA logic blocks to be evaluated in a rigorous scientific way.

1. INTRODUCTION

Field-Programmable Gate Arrays (FPGAs) are reconfigurable integrated circuits that are characterized by a sea of programmable logic blocks (PLBs) surrounded by a programmable routing structure. Most modern PLBs are based on the K-input lookup table (K-LUT) where a K-LUT contains 2^K truth table configuration bits so it can implement any K-input function. Figure 1 illustrates a 2-LUT. There

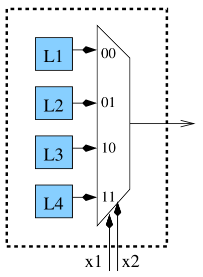


Figure 1: An example 2-LUT.

has been a great deal of research exploring both PLB [4, 6, 10] and interconnect structures for FPGAs [3]. Most of this literature has shown the 4-LUT to be the "best" PLB in terms of balancing area efficiency and realizing good timing performance.

To our knowledge, all previous works on PLB architectures for FPGAs have fallen into one of the three categories below:

- A K-LUT structure is assumed as the basic configurable element with experiments evaluating area and delay tradeoffs for various values of K [10].
- A specialized PLB is proposed and a customized mapping algorithm is implemented to map benchmark circuits using the proposed element [6].
- Specialized Boolean matching techniques are developed to decompose a logic function in such a way so that it matches the structure of the proposed PLB [4].

A limitation of these approaches is their lack of generality. To search for the optimal FPGA PLB, we must be able to answer the following fundamental question: Given a PLB with an arbitrary structure, how beneficial is this PLB for implementing circuits? The method for answering this question must be general and automated so that we can quickly examine a number of different PLBs. In our work, we present a tool named **SATMAP** (satisfiability mapping) that will answer this question.

The rest of this paper is organized as follows. Section 2 provides some brief background information on existing K-LUT techmappers. Section 3 describes our PLB evaluation technique. In Section 4 and 5, we show a detailed description of the **SATMAP** algorithm. Section 6 shows results of the **SATMAP** algorithm for evaluating a number of different PLB structures. Section 7 describes the scalability issues of the **SATMAP** algorithm and possible solutions. Finally, we provide concluding remarks and describe our future work in this area.

2. TECHNOLOGY MAPPING

The technology mapping step in the FPGA CAD flow converts a gate-level network consisting of primitive gates into the PLBs that are present in the target FPGA architecture. The goal of the technology mapping step is to reduce area, delay, or a combination thereof in the network of PLBs that is produced. In this work, delay is proportional to the *depth* of a circuit where the depth of a node is defined as is the longest path from the node to a primary input. A node can be thought as any primitive element in the circuit network, such as a logic gate, and a primary input is a node that has no inputs from other nodes in the network, such as an input pin.

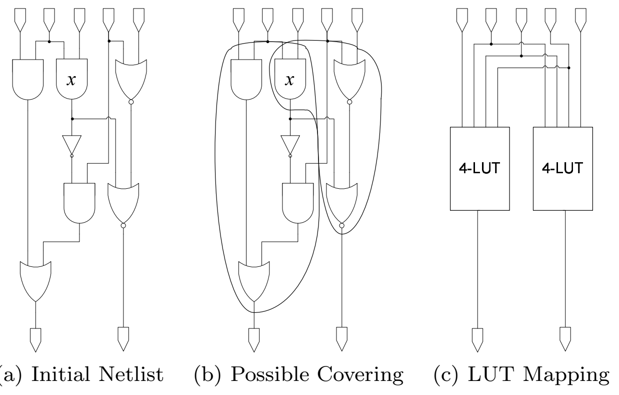


Figure 2: Technology mapping as a covering problem.

The process of technology mapping is often treated as a covering problem. For example, consider the process of mapping a circuit into LUTs as illustrated in Figure 2. Figure 2a illustrates the initial gate level network, Figure 2b illustrates a possible covering of the initial network using 4-LUTs, and Figure 2c illustrates the LUT network produced by the covering. In the mapping given, the gate labeled x is covered by both LUTs and is said to be *duplicated*. Techniques that map for depth use large amounts of duplication to obtain solutions of reduced depth while techniques that map for area limit the use of duplication as it often increases the area of the mapped solutions.

2.1 Technology Mapping for LUTs

There has been a large body of work exploring technology mapping for LUTs. This work is still applicable for PLBs, so long as heuristics are used to ensure the final decomposition of the circuit will map to the PLB architecture.

One of the earliest works to study the depth minimization problem in LUT mapping showed that the depth-optimal mapping solution can be obtained in polynomial time using a dynamic programming procedure [13].

In contrast to the depth minimization problem, the area minimization problem was shown to be NP-hard for LUTs of size four and greater [15]. Thus, heuristics are necessary to solve the area minimization problem.

Early work considered the decomposition of circuits into a set of trees which were then mapped for area [16]. The area minimization problem for trees is much simpler and can be solved optimally using dynamic programming. However, real circuits are rarely structured as trees and tree decomposition prevents much of the optimization that can take place across tree boundaries.

In a *duplication-free* mapping, each gate in the initial circuit is covered by a single LUT in the mapped circuit. The area minimization problem in duplication-free mapping can be solved optimally by decomposing the circuit into a set of maximum fanout free cones (MFFCs) which are then mapped for area [14]. Although the area minimal duplication-free mapping is significantly smaller than the area minimal tree mapping, the controlled use of duplication can lead to further area savings. In [17], heuristics are used to mark a set of gates as *duplicable.* Then area optimization is considered within an extended fanout free cone (EFFC) where an EFFC is an MFFC that has been extended to include duplicable gates.

3. PLB ROBUSTNESS

Another way to look at technology mapping is as a *cone* selection problem. A cone is defined as the network or subgraph formed by taking a root node, v, and some of its predecessors, u, such that for every node u in the cone there exists a path from u to v that lies entirely in the cone. The subcircuits highlighted in Figure 2b are a examples of cones.

Technology mapping seeks to find the best set of cones that can be mapped to the current PLB architecture. "Best" is determined by the optimizing goal, such as area, speed, or power. If the FPGA architecture consists solely of K-LUTs, mapping from cones to K-LUTs is a direct process since any cone with K-inputs or less can be implemented in a K-LUT. A cone with K-inputs or less is known to be K-*feasible*. Thus, the circuit simply has to be decomposed into a set of K-feasible cones. However, if the FPGA architecture consists of generic K-input PLBs, mapping from cones to PLBs is much more difficult since PLBs cannot implement all possible K-feasible cones. For example, consider the PLB shown in left and side of Figure 3, which consists of a 2-LUT feeding an `OR` gate. An example cone that cannot be implemented in this PLB is a 3-input `AND` gate shown in the right hand side of Figure 3. Due to this limitation, when technology mapping to generic PLBs, heuristics are necessary to either decompose the circuit such that it can directly map to the PLB architecture or have a selection process that can discard cones that cannot map into the PLB.

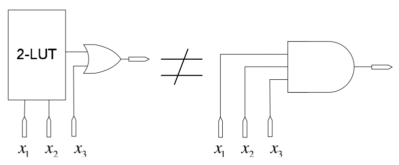


Figure 3: PLB limitations example.

Although PLBs seem much more limited then fully programmable K-LUTs, programmable logic is expensive in terms of area, speed, and power. Thus, PLB non-programmable components can be useful in many cases. Furthermore, in general only a small subset of K-feasible cones will appear in most logic circuits. Thus, so long as the PLB can capture most K-feasible cones that occur in real circuits, the benefits of having non-programmable logic will outweigh the area, power, and speed costs of solely using LUTs. PLB designers come up with clever PLB designs to accomplish this. One way to aid in this design process is to extract a set of K-feasible cones from a set of circuits and determine how many of these cones can fit into a given PLB. A high fit percentage implies that the non-programmable components of the PLB are usually not going to waste and the PLB architecture will be useful. The tool **SATMAP** does this exact task, which will be described in the following section.

4. AN OVERVIEW OF SATMAP

SATMAP has two modes of operation: technology mapping and PLB evaluating. In both modes, **SATMAP** reads in a description of the PLB architecture. As a technology mapper, **SATMAP** finds a feasible circuit mapping to that architecture; and as a PLB evaluator, **SATMAP** will determine a cone fit percentage when given a set of circuits.

4.1 PLB Technology Mapper

```
1  GENERATECONES()
2  REMOVENOFITCONES()
3  for   $i \leftarrow 1$ upto $MaxI$
4        TRAVERSEFWD()
5        TRAVERSEBWD()
6  end for
7  CONESTOPLBS()
```

Figure 4: A overview of the SATMAP algorithm.

We base our work on IMap [8], an iterative K-LUT technology mapping algorithm. For a detailed description of IMap please refer to [8], which shows that IMap produces amongst the best area results of any known technology mapper. The basic framework for the **SATMAP** algorithm is presented in Figure 4. First, a call to GENERATECONES generates a subset of most K-feasible cones for each node in the graph, where K is the input size of the PLB. Next, a call to REMOVENOFITCONES discards all cones that cannot fit into the PLB architecture. This decision process uses SAT and will be described in the following section. Once a set of valid cones is found, a series of forward and backward graph traversals is started to select the best cover of the graph. The cost of the cover is measured in terms of area and depth. The forward traversal, TRAVERSEFWD, selects a cone for each node, and the backward traversal, TRAVERSEBWD, selects a set of cones to cover the graph. Iteration is beneficial because every backward traversal influences the behavior of the forward traversal that follows it.

During the forward traversal, the algorithm updates the depth and the *area flow* for every node and edge encountered. Area flow is a heuristic for estimating the area of the mapping solution below a node or an edge where minimizing it leads to smaller mapping solutions [8]. At each internal node v, a cone rooted at v is selected to cover v and some of its predecessors in a mapping solution. The quality of the mapping solution is determined by the selection procedure and thus the set of cones selected.

During depth-oriented mapping, on the first mapping iteration, the cone with the lowest depth is selected. The first forward traversal establishes the optimal mapping depth, *ODepth*, which can then be used in subsequent iterations to bound the depth of cones selected at every node. Using the optimal depth and the *height* of a node v, a bound can be defined on the depth of a cone C_v as follows

$$depth(C_v) \leq ODepth - height(v). \quad (1)$$

The height of a node or cone is defined as the longest path from that node or cone to a primary output of the circuit. A primary output is any node that does not feed anything else in the circuit, such as a output pin. Cones that meet the bound requirement are preferred and among a set of cones that meet the bound requirement, cones with lower area flows are selected. This selection strategy ensures that the mapping solutions will still achieve the optimal depth selected while minimizing area.

During area-oriented mapping, the cone with the lowest area flow is selected and if cones are equivalent in area flow, then the one with the lowest depth is selected.

During the backward traversal, internal nodes of the graph are visited in the reverse topological where a cover of cones is produced. During this traversal, the $height(v)$ of all internal nodes are updated to the height of the cone covering it. This is for use in Equation 1 in the next forward traversal. If v is found in several cones, the largest height is used.

Finally, a call to CONESTOPLBS converts the cones selected by the final backward traversal into PLBs.

4.1.1 *Generating k-Feasible Cones*

A version of the algorithm described in [17] is used to generate and store all K-feasible cones in the graph. The K-feasible cones are generated as the graph is traversed in topological order from primary inputs to primary outputs. At every internal node v, new cones are generated by combining the cones at the input nodes. In contrast to the original IMap algorithm which combined the cones in every possible way, in our work, the cone generation algorithm combines cones if they have no more $(k + e)$ inputs in total. As long as e was set to a sufficiently high number (2 in the experiments), this heuristic sped up the cone generation process without significantly impacting the quality of the mapping solution.

4.2 PLB Evaluate

```
1  $X \leftarrow$ GENERATECONES()
2  $Y \leftarrow$ REMOVENOFITCONES()
3  $FitPercent \leftarrow (X - Y)/X$
```

Figure 5: A overview of the PLB Evaluator.

As stated previously, PLBs that can capture the functionality of most cones found in real circuits are desired since their non-programmable components will not be wasted. In order to help find such PLBs, **SATMAP** can be used to return a PLB cone fit percentage where a high fit percentage is preferred. This fit percentage can be found by taking a subset of steps from the technology mapping algorithm. In PLB evaluate mode, **SATMAP** generates a set of K-feasible cones extracted from a list of circuits given to it. Next, it removes the set of cones that cannot fit into the current PLB architecture using our SAT based decision process. Recording the number of cones generated and discared, a fit percentage for various PLB architectures can be found as shown in Figure 5, line 3.

5. CONE FITTING USING SAT

In order to determine if a cone can fit into a PLB, we adapt a technique used previously in [19]. In [19], the author formulates the cone fitting problem as a SAT problem. The SAT problem seeks an assignment to the variables of a Boolean function, F, such that F will evaluate to true. If this assignment is possible, F is said to be *satisfiable.* SAT solvers are tools used to solve the SAT problem [9]. In general, SAT solvers only work on Boolean functions expressed in Conjunctive-Normal-Form (CNF), where the expression consists of a conjunction of clauses and each clause consists of a disjunction of literals. Figure 6 is an example Boolean function in CNF. In CNF, the SAT problem seeks an assignment to the variables of F such that each clause has at least one literal evaluating to true. When relating this to our cone fitting problem, a satisfiable assignment implies that the cone will fit into the PLB architecture, otherwise, a fit is not possible.

$$F = (x_1 + \underbrace{\overline{x_2}}_{\text{literal}}) \cdot \underbrace{(\overline{x_2} + x_3 + \overline{x_4})}_{\text{clause}} \cdot (x_2)$$

Figure 6: An example Boolean function in CNF.

A detailed derivation of converting our cone fitting problem into a SAT problem is shown in [19]. We use an example to describe the basic mechanism for determining if a K-feasible cone can be implemented in a general K-input PLB.

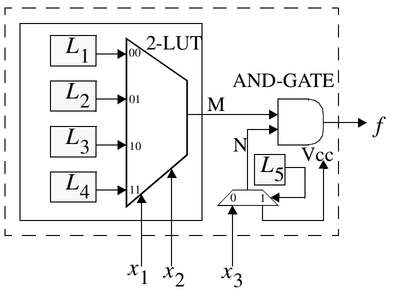


Figure 7: Example PLB.

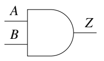
Figure 7 shows an example of a PLB that is more complicated than a simple LUT. It consists of a 2-LUT followed by an `AND` gate (2-LUTAND). The bits $L_1 \dots L_5$ represent the configuration bits for this PLB.

The function of this PLB $f(x_1, x_2, x_3)$ can be described in terms of its configuration bits $(L_1 \dots L_5)$, inputs $(x_1x_2x_3)$, output (f), and wire connections (M, N) to form a characteristic function. The characteristic function describes a valid input-output behaviour of the circuit. Thus, by setting the output variables to match the function in question, a satisfiable assignment will imply that there is a valid set of input vectors and configuration bits to realize this function. For small SAT instances, modern SAT solvers (Chaff [9]) can be used to efficiently solve the problem. In fact, these solvers are routinely used on SAT problems that have thousands of Boolean variables.

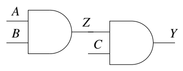
5.1 Constructing CNF Expression

To derive the characteristic function of the current PLB architecture, we use a conversion process as described in [7]. This produces a CNF formula which describes all valid inputs, output, configuration bits, and internal signal vectors. For example, consider Figure 8. It shows the characteristic function for an `AND` gate, followed by the characteristic function of two chained `AND` gates.

Note that Equations 2 and 3 will be satisfied if and only if the variables representing each input, output, and internal wires hold a feasible logic representation. For the single `AND` gate, the CNF can only be satisfied for input and output combinations that are consistent with the `AND` function (e.g. $(A = 0, B = X, Z = 0)$, $(A = X, B = 0, Z = 0)$ or



$$F_1(A, B, Z) = (A + \overline{Z}) \cdot (B + \overline{Z}) \cdot (\overline{A} + \overline{B} + Z) \qquad (2)$$



$$\begin{aligned} F_2(A, B, C, Z, Y) = &(A + \overline{Z}) \cdot (B + \overline{Z}) \cdot (\overline{A} + \overline{B} + Z) \\ &\cdot (Z + \overline{Y}) \cdot (C + \overline{Y}) \cdot (\overline{Z} + \overline{C} + Y) \end{aligned} \qquad (3)$$

Figure 8: `AND` gate characteristic functions.

$(A = 1, B = 1, Z = 1)$. Similarly, the CNF for the cascaded `AND` structure can only be satisfied with values such as $(A = 1, B = 1, C = 0, Z = 1, Y = 0)$. [7] and [19] have a more detailed explanation on how to derive these logic circuit characteristic functions. A PLB characteristic function can be used for PLB legality checking by first constraining the input and output variables according to a given cone truth table under consideration. Next, the CNF formula is run through a SAT solver to determine if there is a valid combination of configuration bits and intermediate signals that can achieve the combination of inputs and outputs. For example, let us consider input $\boldsymbol{C}$ of Figure 8 to be a configuration bit. To check if the constraint $ABY = 111$ is feasible, we attempt to find if $F_2(A = 1, B = 1, C, Z, Y = 1)$ is satisfiable. Clearly, this will return true with $C = 1$.

The previous example only shows that $Y = 1$ is possible if $AB = 11$. This does not determine if all possible assignments to AB can yield a given 2-input function. To check if a 2-input function can be implemented with our simple circuit, all assignments to AB need to be explored. This can be thought as adding a universal quantifier to the variables AB. In informal terms, function fitting is the problem of determining if there exists a configuration to C such that for all values to AB, output Y is equivalent to some function f. Stating the cone fitting problem in this way forms a quantified Boolean formula (QBF) as shown in Figure 9 [1]. To model this such that the problem still can be solved us-

$$\exists C \; \forall AB (Y \equiv f) \qquad (4)$$

Figure 9: QBF representation of cone fitting problem.

ing SAT, its CNF is replicated 2^n times to form a new CNF equation, where n represents the input size of the cone being mapped. Each replecant of the basic PLB CNF equation represents one entry in the cone's truth table (i.e. one possible assignment to variables AB). SAT is performed on the new CNF formula to check if the function can fit into the PLB where a satisfying assignment implies that there exists an assignment to the configuration variables such that the PLB can implement a given function f.

A detailed example of this procedure using the PLB shown in Figure 7 follows. Notice the internal wires have been marked with variable names (M, N) for the purpose of CNF construction. In the following steps, f represents the function of the cone under consideration for mapping, $\boldsymbol{X_i}$ rep-

[1]The full expression is $\exists C \; \forall AB \; \exists Z(Y \equiv f)$; innermost existential quantifiers do not to be explicitly shown.

i	$\boldsymbol{X_i}$	f_i
0	000	0
1	001	0
2	010	0
3	011	0
4	100	0
5	101	0
6	110	0
7	111	1

Table 1: Truth Table describing f.

resents input vector $x_1x_2x_3 = i$, and $f_i = f(\boldsymbol{X_i})$.
Step 1: Create CNF for individual elements in the PLB.

$$\begin{aligned} G_{LUT} = \; & (x_1 + x_2 + \overline{L_1} + M) \cdot (x_1 + x_2 + L_1 + \overline{M}) \\ & \cdot (x_1 + \overline{x_2} + \overline{L_2} + M) \cdot (x_1 + \overline{x_2} + L_2 + \overline{M}) \\ & \cdot (\overline{x_1} + x_2 + \overline{L_3} + M) \cdot (\overline{x_1} + x_2 + L_3 + \overline{M}) \\ & \cdot (\overline{x_1} + \overline{x_2} + \overline{L_4} + M) \cdot (\overline{x_1} + \overline{x_2} + L_4 + \overline{M}) \end{aligned} \tag{5}$$

$$G_{MUX} = (L_5 + \overline{x_3} + N) \cdot (L_5 + x_3 + \overline{N}) \cdot (\overline{L_5} + N) \tag{6}$$

$$G_{AND} = (M + \overline{f}) \cdot (N + \overline{f}) \cdot (\overline{M} + \overline{N} + f) \tag{7}$$

Step 2: Formulate the PLB CNF from Equations 5, 6, and 7. Note that Equation 8 is an expression dependent on the PLB inputs and output (x_{1-3},f), intermediate wire variables (M, N), and configuration variables (L_{1-5}).

$$G_{PLB}(\boldsymbol{X_i}, f_i) = G_{LUT} \cdot G_{MUX} \cdot G_{AND} \tag{8}$$

Step 3: Replication of Equation 8 and constrain inputs and output according to function $f(\boldsymbol{X})$.

$$\begin{aligned} G_{Total} = \; & G_{PLB}(\boldsymbol{X_0}, f_0) \cdot G_{PLB}(\boldsymbol{X_1}, f_1) \\ & \cdot G_{PLB}(\boldsymbol{X_2}, f_2) \cdot G_{PLB}(\boldsymbol{X_3}, f_3) \\ & \cdot G_{PLB}(\boldsymbol{X_4}, f_4) \cdot G_{PLB}(\boldsymbol{X_5}, f_5) \\ & \cdot G_{PLB}(\boldsymbol{X_6}, f_6) \cdot G_{PLB}(\boldsymbol{X_7}, f_7) \end{aligned} \tag{9}$$

Although not explicitly shown in Equation 9, the configuration bits are represented by the same variables (L_{1-5}) in each $G_{PLB}(\boldsymbol{X_i}, f_i)$ instance, where as all other signals (M, N) are replaced by unique variables in each instance. This preserves the meaning of the existential quantifier applied to the configuration bits when in QBF form. The existential quantifier ensures only one configuration will be returned. As a last step, Equation 9 is passed into a SAT solver which will find a satisfying assignment if the cone fits in the PLB structure.

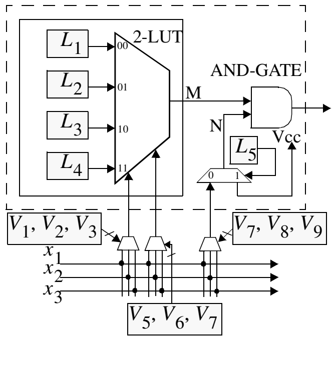


Figure 10: Virtual Multiplexers.

Finally, we note that the pins in the previous example PLB are not permutable. Given the labeling convention in Figure 7, the function $f = (x_1+x_2)\cdot x_3$ can be implemented, but the function $f = (x_1 + x_3) \cdot x_2$ cannot. However most FPGA routing structures are flexible, which allow the input labels in Figure 7 to be permuted. To model this, we use virtual multiplexers controlled by virtual configuration bits $V_1 \ldots V_9$, as shown in Figure 10. Thus, if we wished to map $f = (x_1 + x_3) \cdot x_2$ into this network then the virtual multiplexers would force x_1 and x_3 onto the first two pins of the 2-LUTAND and x_2 to the third pin feeding the 2-AND gate to generate a satisfiable solution. Adding virtual multiplexors can be easily incorporated into the previous example by first adding the virtual multiplexer characteristic functions to **Step 1** then following the rest of the steps as shown previously.

6. RESULTS

6.1 Evaluation of Various PLBs

To show the power of the **SATMAP** algorithm, several unrelated PLB architectures were evaluated. Figure 11 shows the five different PLB architectures used for evaluation. PLB (a) is derived from Altera's Apex20k architecture [2], where as the other structures are derived for research purposes only. To evaluate the versitility of each

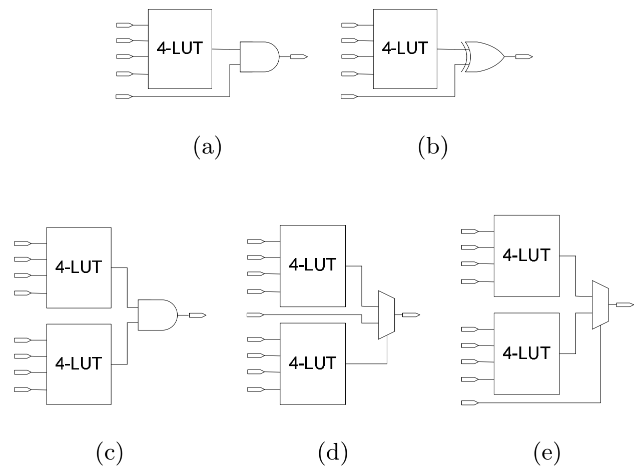


Figure 11: PLB architectures.

PLB, a set of cones were extracted from a list of circuits taken from the MCNC benchmark suite [20] (approximately 1000 K-input cones per circuit, where K was the input size of the PLB). These cones were tested for PLB fitting using the Chaff [9] SAT solver. The circuits used were unrelated to generate a large set of dissimilar cones. Table 2 shows the PLB fit percentage of cones per circuit. The last row shows the total percentage of all cones that fit. Note that the cone fit percentage varies wildly for all PLBs depending on the circuit. This shows that PLB usefulness is dependent on the application of the circuit. Interestingly, PLB (b) failed for all circuits except the ALU circuit (C2670). A reason for this is because PLB (b) uses an XOR gate and XOR gates are

Circuit	*a*	*b*	*c*	*d*	*e*
C2670	27.9	1.59	41.8	0.00	0.00
ex5p	91.4	0.00	49.7	0.00	0.00
clma	61.5	0.00	40.5	1.29	1.29
dalu	78.2	0.00	38.5	0.00	0.00
des	12.2	0.00	72.6	0.00	0.00
i9	87.4	0.00	18.8	0.00	0.00
x3	21.5	0.00	38.9	20.2	20.1
f51m	21.7	0.00	18.0	0.00	0.00
misex3	70.2	0.00	45.4	11.8	12.9
mm30a	20.8	0.00	0.20	0.00	0.00
mult16b	2.91	0.00	0.00	0.00	0.00
% Fit	46.0	0.151	36.4	3.34	3.44

Table 2: Percentage of Cones that Fit into PLB.

very rare in most control circuits and are generally used for arithmetic logic.

6.2 Technology Mapping to Apex20k PLB

The results of the **SATMAP** technology mapper are shown here. The PLB of choice was the 5-input Altera Apex20k PLB. A simplified view, shown in Figure 11a, is used in our experiments. Although a simplified PLB was used, routing constraints were maintained. These constraints required the PLB AND-input to come from an adjacent PLB as illustrated in Figure 12. This can be thought as allowing at most one fanout per PLB to enter a PLB AND-input. This constraint disallows many situations such as a PLB fanout to multiple PLB AND-inputs or an IO-pin that directly feeds a PLB AND-input.

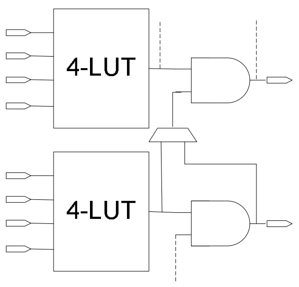


Figure 12: Apex20k PLB Routing Constraints

Table 3 and Table 4 show the total area and depth costs for the largest 20 MCNC benchmark circuits [20]. In Table 3 depth was the primary optimizing goal and in Table 4 area was the primary optimizing goal. For depth the unit delay model is used and for area cost each 4-LUT is given a cost of 1. Area cost uses the assumption that the area of an AND gate is insignificant compared to the area of a 4-LUT. The *Ratio* is the *Total* ratio when compared against **SATMAP** for each respective goal.

The results clearly show that **SATMAP** is an effective tool for technology mapping directly to PLBs as it outperforms any 4-LUT technology mapper. The results are not surprising, since Table 2 shows that the Altera Apex20k PLB can fit a large percentage of cones for any circuit.

7. SCALABILITY ISSUES

Profiling has shown that **SATMAP** spends most of its time generating cones [8] and running SAT. Generating cones takes approximately 90% of the running time for 5-input PLBs, while on top of that, each 5-input cone takes approximately 0.01 to 1 second to evaluate using the Chaff SAT solver [9]. Furthermore, when applied to technology mapping, this technique is not scalable to PLBs with 7-inputs or more. Heuristics in both of these areas are necessary if any technology mapping should occur for larger PLBs.

7.1 Generating Cones

To reduce the runtime of generating cones, a non-exhaustive approach must be taken. One such approach involves a top-down method where a cone is grown downwards starting from the root node. This contrasts with the current bottom up approach we use, which is described in [17]. This could effectively make cone generation constant time with respect to the number of nodes in the network. Heuristics such as area flow would be used to find cones that minimize area costs. An added benefit of reducing the number of cones is that SAT would be applied to a much smaller set of cones, further improving runtime.

7.2 Hardware Acceleration

For the large PLBs evaluated such as Figure 11e, some cones had SAT running times of more than 10 seconds which is not practical for technology mapping[2]. Previous work on Hardware Accelerated SAT solvers can be applied to SAT for up to 80 times speedups [1]. Note that the SAT solvers created in [1] are general SAT solvers, thus their run times include compilation time of the CNF formulae. Work by Zhong et al. [11] shows that compilation time has a significant effect on overall runtime. Since **SATMAP** is aware of the PLB structure before it begins, it can cut out compilation of the CNF formula further increasing the speedup.

8. CONCLUSIONS AND FUTURE WORK

This paper has presented a method for evaluating the efficiency of an arbitrary FPGA PLB. We have shown how our techniques can be applied to several existing FPGA logic structures. In particular, the **SATMAP** algorithm was able to produce significant area savings using the Apex20k PLB structure that contains a 4-input lookup table followed by a 2-input AND gate.

This work represents only the first step in the search for the optimal FPGA PLB. Our research will progress in two distinct areas. The first is hardware acceleration for **SATMAP**. SAT is NP-complete and thus takes a large amount of computation time even with advanced heuristics. The use of specialized hardware acceleration circuitry should speed up our technology mapping time by an order of magnitude or more for very complex PLB structures.

Our second area of research involves an automatic method of chosing the FPGA PLBs to explore. We are exploring a genetic algorithm that can create candidate PLBs from primitive elements such as lookup tables, basic gates, multiplexers and adder structures.

Finally, we must note that the choice of PLB cannot be explored in complete isolation from routing architectures. Issues such as PLB-pin permutability can have a significant

[2]Experiments run on a Sunblade 150 with 1.5GB of RAM.

	Depth Orientated $k = 4$							
	SATMAP		*IMap*		*FlowMap-r0*		*ZMap*	
Circuit	Area	Depth	Area	Depth	Area	Depth	Area	Depth
`alu4`	1003	7	1045	7	1244	7	1204	7
`apex2`	1173	8	1236	8	1468	8	1400	8
`apex4`	997	7	1131	6	1131	6	1113	6
`bigkey`	1145	4	1586	3	1362	3	1698	3
`C6288`	814	24	1023	26	539	25	546	25
`clma`	4689	15	5032	14	5359	15	5297	15
`des`	1128	6	1239	6	1522	6	1359	6
`diffeq`	904	13	1025	12	1420	12	1013	12
`dsip`	1367	4	1144	4	1591	4	1144	4
`elliptic`	2094	16	2239	16	3560	16	2708	16
`ex1010`	2180	8	2639	8	2684	8	2657	8
`ex5p`	983	6	991	7	1055	7	1051	7
`frisc`	2275	21	2397	21	3396	21	2858	21
`i10`	811	14	914	15	953	14	867	14
`misex3`	1117	6	1115	7	1293	7	1244	7
`pdc`	1829	10	2180	10	2216	10	2147	10
`s38417`	4228	10	4296	10	3992	10	3720	10
`s38584.1`	3963	10	4124	11	4437	10	4176	10
`seq`	1102	6	1120	7	1385	7	1292	6
`spla`	1271	8	1380	8	1612	8	1549	8
Total	35073	203	37856	206	42219	204	39043	203
Ratio	1.000	1.000	1.079	1.015	1.204	1.005	1.113	1.000

Table 3: Comparing SATMAP to IMap, FlowMap-r0, and ZMap when performing depth-oriented mapping with $k = 4$.

	Area Orientated $k = 4$							
	SATMAP		*IMap*		*FlowMap-r3*		*ZMap*	
Circuit	Area	Depth	Area	Depth	Area	Depth	Area	Depth
`alu4`	1002	9	1020	9	1144	9	1129	11
`apex2`	1222	13	1173	11	1290	10	1308	13
`apex4`	978	9	1011	10	1099	8	1115	9
`bigkey`	920	5	1034	4	1254	4	1145	4
`C6288`	765	34	972	44	549	25	562	27
`clma`	4261	22	4476	20	4950	17	5014	26
`des`	1103	10	1161	9	1237	8	1194	9
`diffeq`	921	18	1049	16	931	14	941	14
`dsip`	1146	5	1144	4	1145	4	1367	5
`elliptic`	2009	22	2204	23	2133	18	2342	22
`ex1010`	1995	13	2138	14	2397	11	2325	14
`ex5p`	906	11	923	10	956	9	993	11
`frisc`	2152	29	2353	33	2659	24	2624	29
`i10`	785	23	899	22	774	17	779	21
`misex3`	1062	10	1078	9	1185	9	1184	10
`pdc`	1773	15	1755	17	1907	12	1928	17
`s38417`	4039	15	4084	13	3803	12	3586	14
`s38584.1`	3929	14	4018	16	3921	12	3762	16
`seq`	1109	10	1083	11	1204	9	1182	11
`spla`	1287	11	1284	12	1412	11	1403	12
Total	33364	298	34859	307	35950	243	35883	295
Ratio	1.000	1.000	1.045	1.030	1.078	0.815	1.076	0.990

Table 4: Comparing SATMAP to IMap, FlowMap-r3, and ZMap when performing area-oriented mapping with $k = 4$.

impact on the overall area of the FPGA even if we are able to reduce the total number of PLBs required to map a circuit.

9. REFERENCES

[1] M. Abramovici and J. T. de Sousa, “A SAT solver using reconfigurable hardware and virtual logic”, *Journal of Automated Reasoning*, 24(1/2), 2000, pp. 5-36.

[2] Altera Inc. *APEX 20K Data Sheet*, March 2004.

[3] V. Betz and J. Rose, “Effect of the Prefabricated Routing Track Distribution on FPGA Area-Efficiency”, *IEEE Transactions on VLSI*, Vol. 6, No. 3, September 1998, pp. 445-456.

[4] J. Cong and Y.-Y. Hwang, “Boolean Matching for LUT-Based Logic Blocks With Applications to Architecture Evaluation and Technology Mapping”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 9, September 2001, pp. 1077-1090.

[5] J. Cong, J. Peck, and Y. Ding, “RASP: A General Logic Synthesis System for SRAM-Based FPGAs”, *FPGA*, 1996, pp. 137-143.

[6] A.Kaviani and S. Brown, “The Hybrid Field Programmable Architecture” *IEEE Design and Test*, April-June 1999, pp. 74-83.

[7] T. Larrabee, “Test Pattern Generation Using Boolean Satisfiability”, *IEEE Transactions on Computer-Aided Design*, 11(1), 1992, pp.6-22

[8] V. Manohararajah, S. D. Brown, and Z. G. Vranesic, “Heuristics for Area Minimization in Lut-Based FPGA Technology mapping”, *International Workshop on Logic and Synthesis (IWLS’04)* , 2004.

[9] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik, “Chaff: Engineering an Efficient SAT solver”, *Proceedings of the 38th Design Automation Conference (DAC’01)*, June 2001, pp 530-535.

[10] J.S. Rose, R.J. Francis, D. Lewis, and P. Chow, “Architecture of Field-Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency”, *IEEE JSSC*, Vol. 25 No. 5, October 1990, pp. 1217-1225.

[11] P. Zhong, M. Martonosi, P. Ashar, and S. Malik, “Accelerating Boolean Satisfiability with Configurable Hardware”, *IEEE Symposium on FPGAs for Custom Computing Machines*, Los Alamitos, CA, 1998, pp. 186-195

[12] B. U. MVSIS Research Group, *MVSIS*, 2003–2004.

[13] J. Cong and Y. Ding, “An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs” *IEEE Transactions on Computer-Aided Design*, Vol. 13, No. 1, January 1994, pp. 1-13.

[14] J. Cong and Y. Ding, “On Area/Depth Tradeoff in LUT-Based FPGA Technology Mapping”, *IEEE Transactions on VLSI Systems*, Vol. 2, No. 2, June 1994, pp. 137-148.

[15] A. Farrahi and M. Sarrafzadeh, “Complexity of the Lookup-Table Minimization Problem for FPGA Technology Mapping”, *IEEE Transactions on Computer-Aided Design*, Vol. 13, No. 11, pp. 1319–1332.

[16] K. Keutzer, “DAGON: Technology Binding and Local Optimization by DAG Matching”, *Proceedings of the Design Automation Conference*, Miami Beach, FL, 1987, pp. 341-347.

[17] J. Cong, C. Wu and Y. Ding, “Cut Ranking and Pruning: Enabling A General and Efficient FPGA Mapping Solution”, *Proceedings of the ACM International Symposium on FPGAs*, Monterey, CA, February 1999, pp. 29-35.

[18] J. Cong and Y-Y. Hwang, “Simultaneous Depth and Area Minimization in LUT-Based FGPA Mapping”, *Proceedings of the ACM International Symposium on FPGAs*, Monterey, CA, February 1995, pp. 68-74.

[19] A. Ling, “Field-Programmable Gate Array Logic Synthesis Using Boolean Satisfiability”, Master’s Thesis, University of Toronto, 2005

[20] S. Yang, “Logic synthesis and optimization benchmarks user guide version”, 1991