FPGA Logic Synthesis using Quantified Boolean Satisfiability

Andrew Ling[1], Deshanand P. Singh[2], and Stephen D. Brown[2]

[1] University of Toronto, Toronto ON, Canada,
aling@eecg.toronto.edu
[2] Altera Corporation, Toronto Technology Centre, Toronto ON, Canada,
{dsingh|sbrown}@altera.com

Abstract. This paper describes a novel Field Programmable Gate Array (FPGA) logic synthesis technique which determines if a logic function can be implemented in a given programmable circuit and describes how this problem can be formalized and solved using Quantified Boolean Satisfiability. This technique is general enough to be applied to any type of logic function and programmable circuit; thus, it has many applications to FPGAs. The applications demonstrated in this paper include FPGA technology mapping and resynthesis where their results show significant FPGA performance improvements.

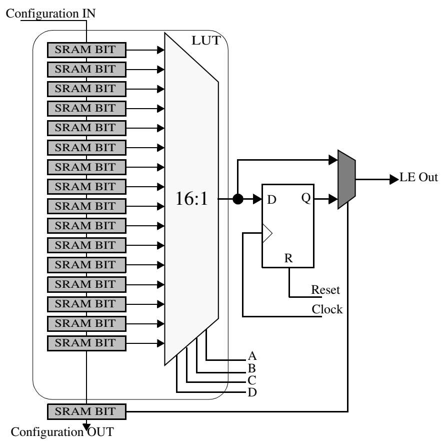
1 Introduction

FPGAs are integrated circuits characterized by two distinct features: programmable logic blocks and programmable interconnect structures. Sec. 1.1 describes a simplified version of the logic block used in most modern FPGA architectures and Sec. 1.2 describes different topologies used to implement the FPGA's programmable interconnect structure.

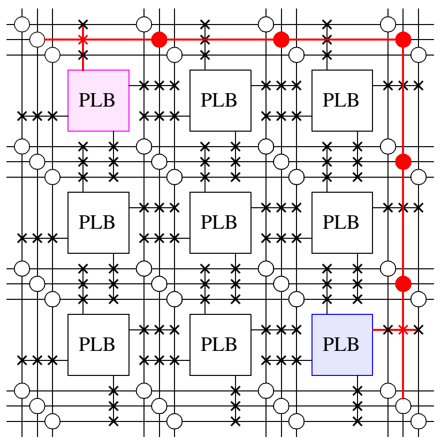
1.1 Programmable Logic Blocks

An example of a programmable logic block (PLB) is shown in Fig. 1(a). The logic block is composed of a 4-input lookup table (4-LUT) that is capable of implementing any arbitrary boolean function of 4 variables. The LUT is implemented with a set of $2^4 = 16$ static RAM (SRAM) bits that are programmed with the truth-table values for the function to be implemented. The 4 inputs (A, B, C, D) feed a multiplexer that selects the appropriate truth-table value from the SRAM bits. Furthermore, Fig. 1(a) shows that the LUT output can either be sent directly to the PLB output, or it can be registered.

In general, many modern PLBs are based on the k-input lookup table (k-LUT) which contains 2^k SRAM bits. Although the k-input LUT is very flexible, it is usually beneficial to add dedicated non-programmable logic to the PLB such as adders and XOR/AND-gates [1, 16]. These features increase the number of functions that can be implemented by a PLB without the power, speed, and area costs associated with programmable logic. However, because this reduces the flexibility of the PLB, optimal mapping of functions to these non-programmable components is difficult.



(a) Programmable Logic Block



(b) Routing Structure

Fig. 1. Simplified FPGA.

1.2 Programmable Interconnect

Fig. 1(b) shows a popular implementation of the programmable interconnect known as the *segmented* interconnect structure [16]. The horizontal and vertical routing channels consist of a set of prefabricated metal segments that can be connected together with programmable routing switches. In Fig. 1(b), all of the metal segments only span the length of one PLB but a wider distribution of segment lengths is typically used in commercial architectures [1, 16]. Connections between PLBs are made by "connecting" the appropriate programmable switches. Fig. 1(b) highlights an example connection between two PLBs.

1.3 Motivation

The cost of implementing a circuit in an FPGA is directly proportional to the number of PLBs required to implement the functionality of the circuit. FPGAs are sold in a number of pre-fabricated sizes. Decreasing the number of PLBs may allow a circuit to be realized in a smaller FPGA. Typical pricing is roughly linear to the number of PLBs in the FPGA device [8].

The technology mapping step of the FPGA CAD flow takes a gate-level representation of a circuit and produces a netlist of PLBs which implements the same functionality as the gate-level description. Technology mapping has a significant impact on the number of PLBs required to realize a particular circuit. In this paper we will show how methods based on Quantified Boolean Satisfiability can be used to significantly improve state-of-the-art technology mapping algorithms.

2 Background

2.1 Technology Mapping

The goal technology mapping step is to reduce area, delay, or a combination thereof in the PLB network that is produced. Often existing literature on technology mapping uses the term *depth* to refer to delay. We also adopt this terminology.

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 Fig. 2. Fig. 2a illustrates the initial gate level network, Fig. 2b illustrates a possible covering of the initial network using 4-input LUTs, and Fig. 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 solution.

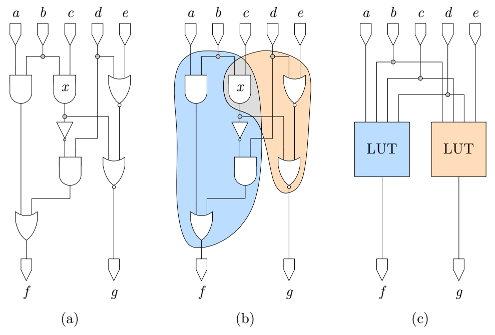


Fig. 2. (a) the initial netlist. (b) possible covering. (c) the mapping produced.

Technology Mapping for LUTs Due to the popularity of LUT-based FPGA architectures [1, 16], a large body of existing work has studied area-driven LUT mapping algorithms. We review some of the key literature here.

The area minimization problem was shown to be NP-hard for LUTs of input size four and greater [9]. Thus, heuristics are necessary to solve the area minimization problem in a reasonable amount of computation time. Early work considered the decomposition of circuits into a set of trees which were then mapped for area [10]. 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 [4]. 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 [6], heuristics are used to mark a set of gates as *duplicable*. Area optimization is then considered within an extended fanout free cone (EFFC) where an EFFC is an MFFC that has been extended to include duplicable gates.

2.2 Terminology

The remainder of this paper uses standard nomenclature for describing FPGA technology mapping algorithms. This is as follows: The combinational portion of a LUT network can be represented as a directed acyclic graph (DAG). A node in the graph represents a LUT, primary input (PI), or primary output (PO). A directed edge in the graph with head u, and tail v, represents a signal in the logic circuit that is an output of node u and an input of node v.

A *cone* of v, C_v, is a subgraph consisting of v and some of its non-PI predecessors such that any node $u \in C_v$ has a path to v that lies entirely in C_v. Node v is referred to as the *root* of the cone. At a cone C_v, the set of input edges is the set of edges with a tail in C_v and the set of output edges is the set of edges with v as a head. The fanin size of a cone is the number of input edges. A cone with n input edges is known to be n-feasible and can be trivially implemented with an n-LUT. A fanout free cone (FFC) is a cone with output edges only originating from the root of the cone. A maximum fanout free cone (MFFC) is a FFC maximizes the number of nodes contained in the FFC.

2.3 Quantified SAT

As stated in Sec. 1, the main contribution of this work is to examine the use of Quantified Boolean Satisfiability for use in FPGA technology mapping. Quantified SAT is the problem of determining if a quantified Boolean formula (QBF), $F = Q_1x_1...Q_nx_nf(x_1...x_n)$ where $Q_i \in \{\exists, \forall\}$, has an assignment to its variables, $x_1...x_n$, such that F evaluates to 1. If so, F is said to be *satisfiable*, otherwise it is *unsatisfiable*. This differs from the SAT problem since SAT seeks a single assignment to its variables to satisfy the Boolean formula. Quantified SAT, however, seeks all assignments to its universally quantified variables to satisfy a QBF. For example, consider the expressions in Equ. 1. The first expression shows a satisfiable Boolean formula with its associated satisfying assignment. In contrast, simply by adding quantifiers to it, the QBF shown in the second expression is unsatisfiable due to the universally quantified variable x_2.

$$\begin{array}{rl} (x_1 + x_2) \cdot (\overline{x_1} + \overline{x_2}) & \text{satisfiable} \rightarrow [x_1 = 0, x_2 = 1] \\ \exists x_1 \forall x_2 \ (x_1 + x_2) \cdot (\overline{x_1} + \overline{x_2}) & \text{unsatisfiable} \rightarrow [x_1 = 0, x_2 = \{0,1\}] \\ & \text{unsatisfiable} \rightarrow [x_1 = 1, x_2 = \{0,1\}] \end{array} \tag{1}$$

For all practical purposes, Quantified SAT only deals with QBFs in Conjunctive-Normal-Form (CNF). Thus, the problem of Quantified SAT can be rephrased to: Given a QBF, $F = Q_1 x_1 ... Q_n x_n f(x_1 ... x_n)$ where $Q_i \in \{\exists, \forall\}$, find an assignment to its variables, $x_1 ... x_n$, such that each clause in $f(x_1 ... x_n)$ has at least one literal that evaluates to 1.

3 Quantified SAT Applied to FPGA Technology Mapping

The goal of FPGA technology mapping is to map logic functions into the FPGA architecture in an optimized way. Although there are many issues encountered when reaching this goal, the underlying question asked to solve FPGA technology mapping is as follows:
Given an n-variable Boolean function, $F_{function}(x_1, x_2, ..., x_n)$, does there exist a programmable configuration to a circuit such that the output of the circuit will equal $F_{function}(x_1, x_2, ..., x_n)$ for all inputs? This question is difficult to answer since FPGAs consist of PLB arrays, which are not able to implement any k-input logic function. Hence robust heuristics are necessary to map logic functions in the FPGA programmable fabric.

3.1 Formalizing FPGA Technology Mapping

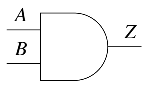
Assuming that a programmable circuit can be represented as a Boolean function $G_{circuit} = G(x_1..x_n, L_1..L_m, z_1..z_o)$ where $x_i, L_j, z_k, G_{circuit}$ represent the input signals, configuration bits, intermediate circuit signals, and output function of the circuit respectively, the problem of function mapping into programmable logic can represented formally as a QBF as follows.

$$\exists L_1 ... L_m \forall x_1 ... x_n \exists z_1 ... z_o (G_{circuit} \equiv F_{function}) \tag{2}$$

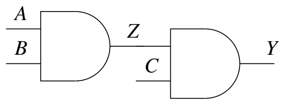
A satisfying assignment to Equ. 2 implies that $F_{function}$ can be realized in the programmable circuit.

In order to derive Equ. 2, the proposition $(G_{circuit} \equiv F_{function})$ must be represented as a CNF Boolean formula. This can be done using a well known derivation technique that converts logic circuits into a *characteristic function* in CNF [11]. This CNF formula describes all valid inputs, output, configuration bits, and internal signal vectors for the configurable circuit. For example, consider Fig. 3. It shows the basic CNF formula for an `AND` gate, followed by the CNF formula of two chained `AND` gates.

Equations 3 and 4 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, this includes all 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 $(A = 1, B = 1, Z = 1)$. Similarly, for the cascaded `AND` structure its characteristic function can only be satisfied with values such as $(A = 1, B = 1, C = 0, Z = 1, Y = 0)$.



$$F_1(A,B,Z) = (A+\overline{Z}) \cdot (B+\overline{Z}) \cdot (\overline{A}+\overline{B}+Z) \tag{3}$$



$$\begin{aligned} F_2(A,B,C,Z,Y) = \; & (A+\overline{Z}) \cdot (B+\overline{Z}) \cdot (\overline{A}+\overline{B}+Z) \\ & \cdot (Z+\overline{Y}) \cdot (C+\overline{Y}) \cdot (\overline{Z}+\overline{C}+Y) \end{aligned} \tag{4}$$

Fig. 3. AND gate CNF formulae

The previous conversion technique for the cascaded `AND` structure can be extended to much larger circuits such as PLBs. This creates a characteristic function, Ψ, dependent on variables $x_1, ..., x_n$, $L_1, ..., L_m$, $z_1, ..., z_o$, and G which represent the inputs, programmable bits, intermediate wires, and output of the circuit respectively. Thus, the proposition $(G_{circuit} \equiv F_{function})$ can be formed by substituting all instances of the output variable G in Ψ by the expression representing $F_{function}$. This is shown in Equ. 5 where the notation $[G/F(x_1, ..., x_n)]$ indicates that all instances of G have been replaced by $F(x_1, ...x_n)$. Sections following this will use similar notation to represent the substitution operation.

$$\begin{aligned} [G_{circuit} \equiv F_{function}] & \equiv \Psi \left[G/F(x_1, ..., x_n)\right] \\ & \equiv \exists L_1...L_m \forall x_1...x_n \exists z_1...z_o \psi \left[G/F(x_1, ..., x_n)\right] \end{aligned} \tag{5}$$

3.2 Removing Quantified Variables

Although Quantified SAT solvers have shown initial promising results, it is often still faster to solve a QBF by removing the universal quantifiers and converting it to a SAT problem [15]. Removing the universal quantifiers eliminates the need to find multiple SAT instances for all universally quantified variable assignments, thus saving time; however, in doing so, the size of the Boolean formula increases substantially. To remove the universal quantifiers in a QBF, F, its proposition, f, is replicated to explicitly enumerate all possible assignments of the universally quantified variables. These replicated formulae are then conjoined with the logical `AND` operator to form a Boolean function that can be solved with SAT.

In order to give better understanding to the previously described ideas, an example is given. Assume that the function listed in Fig. 4 needs to be implemented in the adjacent programmable circuit. The circuit in Fig. 4 consists of a 2-LUT which feeds into a 2-input `AND` gate. In order to test if Fig. 4 can implement a given function, the following steps are taken.

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

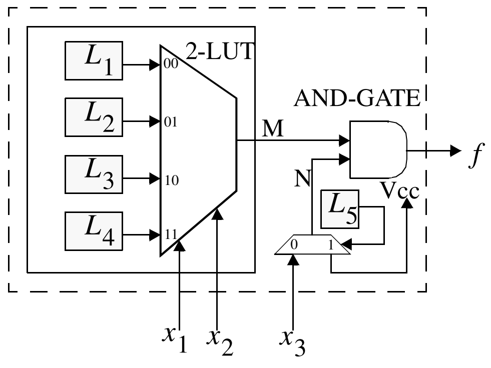


Fig. 4. Example Programmable Circuit

Step 1: Create CNF for individual elements in programmable circuit.

$$\begin{aligned} G_{LUT} = {} & (x_1 + x_2 + \overline{L_1} + z_1) \cdot (x_1 + x_2 + L_1 + \overline{z_1}) \cdot \\ & (x_1 + \overline{x_2} + \overline{L_2} + z_1) \cdot (x_1 + \overline{x_2} + L_2 + \overline{z_1}) \cdot \\ & (\overline{x_1} + x_2 + \overline{L_3} + z_1) \cdot (\overline{x_1} + x_2 + L_3 + \overline{z_1}) \cdot \\ & (\overline{x_1} + \overline{x_2} + \overline{L_4} + z_1) \cdot (\overline{x_1} + \overline{x_2} + L_4 + \overline{z_1}) \end{aligned} \tag{6}$$

$$G_{MUX} = (L_5 + \overline{x_3} + z_2) \cdot (L_5 + x_3 + \overline{z_2}) \cdot (\overline{L_5} + z_2) \tag{7}$$

$$G_{AND} = (z_1 + \overline{G}) \cdot (z_2 + \overline{G}) \cdot (\overline{z_1} + \overline{z_2} + G) \tag{8}$$

Step 2: Formulate the programmable circuit CNF from equations 6, 7, and 8.

$$G_{circuit} = G_{LUT} \cdot G_{MUX} \cdot G_{AND} \tag{9}$$

Step 3: Replication of equation 9 to remove quantified variables. This formulates G_{Total} where a satisfiable assignment to G_{Total} implies F can be realized in the programmable circuit.

$$\begin{aligned} G_{Total} = {} & G_{circuit}[\boldsymbol{X}/\boldsymbol{X_0}, G/F_0, z_1/z_3, z_2/z_4] \cdot G_{circuit}[\boldsymbol{X}/\boldsymbol{X_1}, G/F_1, z_1/z_5, z_2/z_6] \cdot \\ & G_{circuit}[\boldsymbol{X}/\boldsymbol{X_2}, G/F_2, z_1/z_7, z_2/z_8] \cdot G_{circuit}[\boldsymbol{X}/\boldsymbol{X_3}, G/F_3, z_1/z_9, z_2/z_{10}] \cdot \\ & G_{circuit}[\boldsymbol{X}/\boldsymbol{X_4}, G/F_4, z_1/z_{11}, z_2/z_{12}] \cdot G_{circuit}[\boldsymbol{X}/\boldsymbol{X_5}, G/F_5, z_1/z_{13}, z_2/z_{14}] \cdot \\ & G_{circuit}[\boldsymbol{X}/\boldsymbol{X_6}, G/F_6, z_1/z_{15}, z_2/z_{16}] \cdot G_{circuit}[\boldsymbol{X}/\boldsymbol{X_7}, G/F_7, z_1/z_{17}, z_2/z_{18}] \end{aligned} \tag{10}$$

Note that in equation 10, the configuration bits are represented by the same variables (L_{1-5}) in each $G_{circuit}(\boldsymbol{X_i}, f_i)$ instance, where as all other signals are unique variables in each instance. This ensures that only one configuration will exist for all entries of the truth table.

4 Quantified SAT FPGA Applications

4.1 Application 1: General Technology Mapping

The technology mapping problem for general PLBs can be solved using similar techniques described in the Sec. 2.1. However, unlike k-LUTs, not every k-feasible cone can be implemented in a k-input PLB. Thus, a legality check using our Quantified SAT technique must be done before a function is mapped into a k-input PLB. To explore this idea, we developed a tool called **SATMAP** which is a general PLB technology mapper.

```
GenerateCones()
for $i \leftarrow 1$ upto $MaxI$
    TraverseFwd()
    TraverseBwd()
end for
ConesToPLBs()
```

Fig. 5. A high-level overview of the SATMAP algorithm.

An Overview of SATMAP SATMAP is based upon IMap [12], an iterative k-LUT technology mapping algorithm. We present a high level overview of the algorithm here. For a detailed description of IMap please refer to [12], 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 Fig. 5. The key difference between SATMAP and IMap is the function GenerateCones. This generates the set of all k-feasible cones for each node in the graph. In contrast to IMap, SATMAP adds a legality check to every cone where cones that cannot fit into the PLB structure are discarded.

After GenerateCones, 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 edge, where minimizing area flow leads to smaller mapping solutions [12]. 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

$$depth(C_v) \leq ODepth - height(v). \tag{11}$$

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 Equ. 11 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.

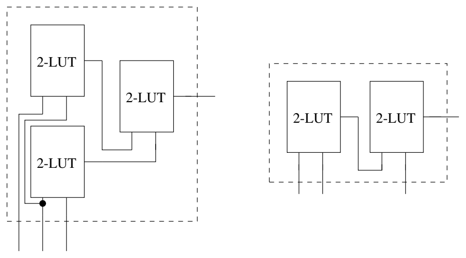
Generating *k*-Feasible Cones A version of the algorithm described in [6] is used to generate and store all k-feasible cones in the graph. The k-feasible cones are generated as the graph is traversed in topological order from PIs to POs. At every internal node v, new cones are generated by combining 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. During the generation, a cone legality check is performed using the function fitting technique described in Sec. 3.1. If a cone does not fit into the PLB architecture, it is discarded, leaving a legal set of cones for the forward and backward traversals of SATMAP.

4.2 Application 2: Area-driven mapping for *k*-LUTs

When mapping a LUT network for area, one must attempt to reduce the number of LUTs in the network yet maintain the original functionality. The more LUTs that can be removed, the farther the original circuit is from the optimal mapping. This can be achieved by resynthesizing smaller subcircuits and applying this in a sliding window fashion over the larger circuit. These subcircuits form a cone, therefore by resynthesizing several cones, this will reduce the LUT count of the overall LUT network. This is simply a logic fitting problem where a logic function is extracted from a cone consisting of X k-input LUTs and then is checked if it can fit into a programmable structure containing less than X k-input LUTs. This can be solved using our Quantified SAT technique.

To illustrate this process, consider Fig. 6. The original cone 6a consists of three 2-LUTs which implements a three input function. Since only three inputs enter the cone, it may be possible to resynthesize 6a into 6b to save one LUT.

To determine if resynthesis from 6a to 6b is possible, 6b is converted into a CNF expression as described in Sec. 3 and the function extracted from 6a is



(a) Original Cone (b) Resynthesized Cone

Fig. 6. Resynthesis of three-input cone of logic.

tested using Quantified SAT to see if it fits into 6b. If the expression is satisfiable, resynthesis can occur.

5 Results

5.1 Technology Mapping to Apex20k PLB

The results of **SATMAP** for PLB technology mapping are shown here. The PLB of choice was the 5-input Altera Apex20k PLB which consists of a 4-LUT feeding into a 2-input `AND` gate. A simplified view, shown in Fig. 7, 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 Fig. 7. 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.

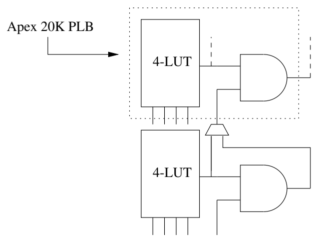


Fig. 7. Two Apex20k PLBs with routing constraints shown.

Tab. 5.1 shows the total depth and area costs for the largest 20 MCNC benchmark circuits where only the 10 largest circuits have been shown in detail. The results clearly show that **SATMAP** [3] is an effective tool for technology

[3] SATMAP was incorporated with the Berkeley MVSIS project [3]

mapping to PLBs as it outperforms any 4-LUT technology mapper. 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 upper table show results when depth was the primary optimizing goal and the lower table show results when area was the primary optimizing goal. *Ratio* show is the *Total* ratio when compared against **SATMAP** for each respective goal.

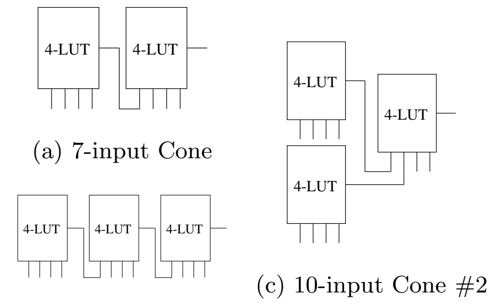
Table 1. Comparing **SATMAP** to IMap, FlowMap-r3, and ZMap with $k = 4$.

	Depth Orientated $k = 4$							
	SATMAP		*IMap*		*FlowMap-r0*		*ZMap*	
Circuit	Area	Depth	Area	Depth	Area	Depth	Area	Depth
`apex2`	1173	8	1236	8	1468	8	1400	8
`clma`	4689	15	5032	14	5359	15	5297	15
`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
`frisc`	2275	21	2397	21	3396	21	2858	21
`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
`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

	Area Orientated $k = 4$							
	SATMAP		*IMap*		*FlowMap-r3*		*ZMap*	
Circuit	Area	Depth	Area	Depth	Area	Depth	Area	Depth
`apex2`	1222	13	1173	11	1290	10	1308	13
`clma`	4261	22	4476	20	4950	17	5014	26
`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
`frisc`	2152	29	2353	33	2659	24	2624	29
`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
`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

5.2 Resynthesis Results

For the resynthesis application, we performed resynthesis on circuits produced by the ZMap techmapper — one of the best publically available FPGA area-driven techmappers developed by J. Cong et al. at UCLA [5]. Given a set of circuits, we used ZMap to technology map these circuits to **4**-LUTs. After some post processing done by RASP [5] to further improve area, we ran our resynthesizer [4] on these LUT networks.



(a) 7-input Cone

(b) 10-input Cone #1

(c) 10-input Cone #2

Fig. 8. Resynthesis Structures

The number of resynthesis structures is countless; however, considering that the size of the CNF equation is exponential to the number of resynthesis structure inputs, for practical purposes, our work dealt with cones of fanin size 10 or less. This limits the number of resynthesis structures to the ones shown in Fig. 8.

Fig. 8a is applied for cones with a fanin size of seven or less and containing more than two 4-LUTs; Fig. 8b and 8c are applied for cones with a fanin size of 10 or less and containing more than three 4-LUTs. Resynthesis checking was done using the Chaff SAT solver developed by M. W. Moskewicz et al. [13].

Benchmark Circuits In our first set of resynthesis experiments, we focused on a set of circuits taken from the MCNC and ITC'99 benchmark suites ([17],[7]). These circuits were optimized using SIS [14] and RASP, technology mapped with ZMap, and resynthesized with our work. The optimization in SIS is particularly important since the structure of the gate-level netlist can have a significant impact on the mapped area. The left hand table in Tab. 5.2 shows the results. The *ZMap* column indicates the number of 4-LUTs the circuit was technology mapped to. The *Resynth* column indicates the number of 4-LUTs after our resynthesis.

The results clearly show that ZMap does not achieve optimal results; this implies that all FPGA techmappers that perform worse then ZMap also have much room for improvement. Notice that the largest decreases in area are seen in circuits with 3000 LUTs or more, with the largest decrease of more than 9% (`s38584.1`). This suggests that the deviation from the optimal solution is

[4] Our resynthesizer was incorporated with the Berkeley MVSIS project [3]

Table 2. Resynthesis results for benchmark circuits and common logic blocks.

Circuit	*Resynth*	*ZMap*	*Ratio*
clma	4792	5014	0.95
b15_1	4112	4291	0.95
b15_1_opt	3772	3879	0.97
s38584.1	3454	3771	0.91
s38417	3444	3586	0.96
b14	2902	3072	0.94
frisc	2571	2624	0.98
pdc	1875	1928	0.97
misex3	1156	1184	0.98
seq	1162	1182	0.98
alu4	1103	1129	0.98
ex5p	968	993	0.97
i10	764	789	0.97
Total	32075	33442	0.96

Building Block	*Resynth*	*ZMap*	*Ratio*
4:1 MUX	2	3	0.67
16:1 MUX	21	29	0.72
32-Bit Priority Encoder	59	74	0.80
4-Bit Barrel Shifter	8	12	0.67
16-Bit Barrel Shifter	32	48	0.67
6-Bit Set Reset Checker	2	3	0.67
2-Bit Sum Compare Const	2	6	0.33
2-Bit Sum Compare	2	3	0.67
6-Bit Priority Checker	3	6	0.50
8-Bit Bus Multiplexor	16	24	0.67
Total	188	253	0.74

proportional to the size of the circuit. The fact that area driven technology mapping is NP-hard [9] supports this claim.

Building Block Circuits In our second set of resynthesis experiments, we focused on common digital circuit logic blocks. We started from Verilog code, synthesized it using VIS [2], then optimized and techmapped the circuits as in the benchmark circuit section. For illustration, Module 1 shows one Verilog circuit description that we synthesized then resynthesized.

Module 1 16-Bit Barrel Shifter Verilog Code

```
module BarrelShifter16Bit(SHIFT,D,Q)
  input[1:0] SHIFT;input[15:0] D;
  output[15:0] Q;reg[15:0] Q;
  always @ (D or Q or SHIFT)
    case (SHIFT)
      2'b00 : Q=D;
      2'b01 : Q={D[3:0],D[15:4]};
      2'b10 : Q={D[7:0],D[15:8]};
      2'b11 : Q={D[11:0],D[15:12]};
    endcase
endmodule
```

The right hand table in Tab. 5.2 shows our results, where we achieve a reduction as large as 67% and an average reduction of 26%. Since these logic blocks are common in digital circuits, heuristics can be used to identify them and technology map them to our optimized circuits.

It is interesting to note the dramatic differences in results between the benchmark circuits and the results of the individual building blocks. We speculate that common building blocks are being collapsed with other random or glue logic in the benchmarks. Since we limit the size of the subcircuit resynthesis procedure, it is likely that we are missing some key resynthesis opportunities.

QBF vs. SAT In order to show the benefits of removing quantifiers in our QBF to produce a SAT problem as shown in Sec. 3.2, we ran a few cone fitting examples using a QBF solver and a SAT solver. The example used a 7-input function realized with two 4-LUTs as in Fig.8a. An unsatisfiable function was selected so the entire search space was explored. Furthermore, the configuration bits were pre-configured to vary the size of the search space. This is shown in Tab. 3. *Config Bits* shows the number of unconfigured programmable bits in the circuit; *SAT* shows the Chaff SAT solver [13] running times on a Sunblade 150 with 2.5 GB of RAM; and *QBF* shows the Quaffle QBF solver [18] running times on the same machine.

Table 3. SAT and QBF solver running times.

Config Bits	SAT (sec)	QBF (sec)
47	0.01	4.16
48	0.38	11.3
49	0.93	45.11
50	1.23	375.12
51	1.75	403.67
52	2.56	1366.66
53	2.70	4117.38

6 Conclusion and Future Work

In this work, we have shown a practical application of Quantified Boolean Satisfiability to the problem of cost reduction of FPGA-based circuit realizations. We have described two different methods:

- A generic technology mapping that is capable of mapping a circuit description to any arbitrary logic block. This allows for the study of architectural features that can be used to reduce the number of PLBs required to implement a circuit. In particular, we have shown that the use of an extra dedicated `AND`-gate can lead to significant area reductions in many cases.
- A method to reduce the number of k-input LUTs required to implement subcircuits for LUT-based FPGA architectures. We have shown area reductions of up to 67% in some cases using these techniques.

In addition to the presentation of QBF applications to FPGAs, we have shown this class of problem that arises in this work is very difficult for QBF solvers. In fact, it seems that a naive translation to SAT is a far better approach than the QBF representation. We hope to provide a number of benchmarks that will help to drive the development of an efficient QBF solver.

References

1. Altera. Component selector guide ver 14.0, 2004.
2. R. K. Brayton and G. D. H. et al. VIS: a system for verification and synthesis. In *Proceedings of the Eighth International Conference on Computer Aided Verification CAV*, pages 428–432, 1996.
3. D. Chai, J. Jiang, Y. Jiang, Y. Li, A. Mishchenko, and R. Brayton. MVSIS 2.0 Programmer's Manual, UC Berkeley. Technical report, Electrical Engineering and Computer Sciences, University of California, Berkeley, 2003.
4. J. Cong and Y. Ding. On area/depth trade-off in LUT-based FPGA technology mapping. In *Design Automation Conference*, pages 213–218, 1993.
5. J. Cong, J. Peck, and Y. Ding. RASP: A general logic synthesis system for SRAM-based FPGAs. In *FPGA*, pages 137–143, 1996.
6. J. Cong, C. Wu, and Y. Ding. Cut ranking and pruning: enabling a general and efficient fpga mapping solution. In *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, pages 29–35. ACM Press, 1999.
7. F. Corno, M. Reorda, and G. Squillero. RT-level ITC 99 benchmarks and first ATPG results, 2000.
8. A. Electronics.
9. A. Farrahi and M. Sarrafzadeh. Complexity of the Lookup-Table Minimization Problem for FPGA Technology Mapping. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(11):1319–1332, 1994.
10. K. Keutzer. Dagon: Technology binding and local optimization by dag matching. In *DAC*, pages 341–347, 1987.
11. T. Larrabee. Test Pattern Generation Using Boolean Satisfiablity. *IEEE Transactions on Computer-Aided Design*, 11(1):6–22, 1992.
12. V. Manohararajah, S. D. Brown, and Z. G. Vranesic. Heuristics for area minimization in lut-based fpga technology mapping. In *International Workshop on Logic and Synthesis (IWLS'04)*, 2004.
13. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, 2001.
14. E. M. Sentovich, K. J. Singh, C. M. L. Lavagno, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical report, Electrical Engineering and Computer Sciences, University of California, Berkeley, 1992.
15. D. Tang, Y. Yu, D. P. Ranjan, and S. Malik. Analysis of search based algorithms for satisfiability of quantified boolean formulas arising from circuit state space diameter problems. In *SAT '04: The Seventh International Conference on Theory and Applications of Satisfiability Testing*, pages 10–13, May 2004.
16. Xilinx. Virtex-ii complete data sheet ver 3.3, 2004.
17. S. Yang. Logic synthesis and optimization benchmarks user guide version, 1991.
18. L. Zhang and S. Malik. Conflict driven learning in a quantified boolean satisfiability solver. In *ICCAD '02: Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 442–449. ACM Press, 2002.