FPGA PLB EVALUATION USING QUANTIFIED BOOLEAN SATISFIABILITY

Andrew C. Ling

Electrical and Computer Engineering
University of Toronto
Toronto, CANADA
email: aling@eecg.toronto.edu

Deshanand P. Singh, Stephen D. Brown

Altera Corporation
Toronto Technology Centre
Toronto, CANADA
email: 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 application demonstrated in this paper is FPGA PLB evaluation where their results show that this tool allows radical new features of FPGA logic blocks to be evaluated in a rigorous scientific way.

1. INTRODUCTION

FPGAs are integrated circuits characterized by a sea of programmable logic blocks (PLBs).

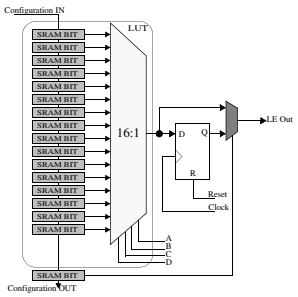


Fig. 1. A simplified FPGA PLB.

An example of a PLB is shown in Fig. 1. 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. 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, 2]. 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.

1.1. 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 [3].

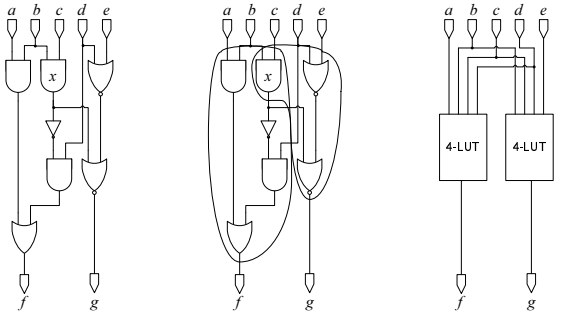
The PLB architecture has a significant impact on the number of PLBs required to realize a particular circuit. Thus, clever PLB designs are necessary that capture the majority of the functions encountered in typical circuits. In this paper we will show how methods based on Quantified Boolean Satisfiability can be used to rapidly determine if a PLB architecture will achieve a high capture rate.

2. BACKGROUND

3. 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 the longest path from the node to a primary input. Previous work showed that the depth-optimal mapping solution can be obtained in

polynomial time using a dynamic programming procedure [4].



(a) Initial Netlist (b) Possible Covering (c) LUT Mapping

Fig. 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 Fig. 2. Fig. 2a illustrates the initial gate level network, Fig. 2b illustrates a possible covering of the initial network using 4-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*. In a *duplication-free* mapping, each gate in the initial circuit is covered by a single LUT in the mapped circuit [5]. However, surprisingly, the controlled use of duplication can lead to further area savings [6]. In contrast to the depth minimization problem, the area minimization problem was shown to be NP-hard for LUTs of size four and greater [7]. Thus, heuristics are necessary to solve the area minimization problem.

Another way to look at technology mapping is as a *cone* selection problem. The subcircuits circled in Fig. 2b are 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, to technology map circuits to K-LUTs, 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, the PLB shown in Fig. 3 cannot implement a 3-input `OR` gate.

Although more limited in functionality, PLBs offer speed, area, and power advantages over fully programmable K-LUTs. Furthermore, in general only a small subset of K-feasible cones will appear in most logic circuits. Thus, so long as a given PLB architecture can capture most cones encountered in real circuits, it will be successful in implementing circuits. One way to determine if a PLB will capture most K-feasible cones found in circuits 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 where a high fit percentage is desired. The tool presented in this paper does this exact task, which will be described in Sec.4.3.

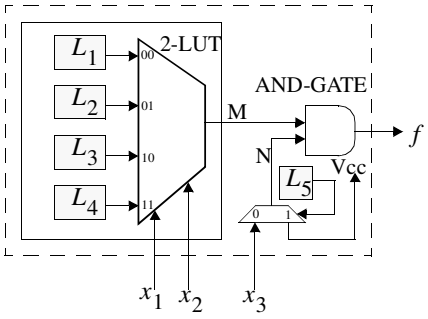


Fig. 3. Example PLB.

3.1. Quantified SAT

As stated in Sec. 1, the main contribution of this work is to examine the use of Quantified Boolean Satisfiability (QSAT) for use in PLB evaluation. QSAT is the problem of determining if a quantified Boolean formula (QBF), $F = Q_1 x_1 ... Q_n x_n f(x_1 ... x_n)$ where $Q_i \in \{\exists, \forall\}$, has an assignment to its variables, $x_1 ... x_n$, such that F evaluates to true. If so, F is said to be *satisfiable*, otherwise it is *unsatisfiable*. This is analogous to the much simplier problem of Boolean Satisfiability (SAT) where SAT seeks a *single* assignment to a Boolean formula F such that F evaluates to true. SAT is actually a special case of QSAT where SAT deals with Boolean formulae without any universal quantifiers (variables in Boolean formulae without any quantifiers implicitly have a single existential quantifier bound to them). QSAT, however, may have universally quantified variables, and thus 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.

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

For all practical purposes, QSAT only deals with QBFs in Conjunctive-Normal-Form (CNF, sometimes referred as

a Product-of-Sums). A Boolean function is in CNF if it consists solely of a conjunction of clauses, where a clause is a disjunction of literals and a literal is any variable or its complement. Equ. 1 are examples of formulae in CNF. In CNF, the problem of QSAT 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 true.

4. QUANTIFIED SAT APPLIED TO PLB EVALUATION

The goal of PLB evaluation is to determine how useful a new PLB architecture will be in implementing circuits. A useful k-input PLB can be characterized by how many k-input cones can fit into it where a high fit percentage is desired. Thus, the underlying question asked when determining this fit percentage 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, G, such that the output of the circuit will equal $F_{function}(x_1, x_2, ..., x_n)$ for all inputs? Previously, robust heuristics to answer this question fell into two categories: a specialized PLB is proposed and a customized mapping algorithm is implemented to map benchmark circuits using the proposed element [8]; 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 [9]. Both of these techniques suffer a lack of generality, which we address in our novel QSAT based approach.

4.1. Formalizing Function Fitting Problem

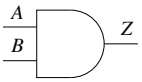
Assuming that a programmable circuit can be represented as a Boolean function $G_{circuit} = G(x_1..x_n, L_1..L_m, z_1..z_o)$ where $x_i, L_j, z_k, G_{circuit}$ represent the input signals, configuration bits, intermediate circuit signals, and output function of the circuit respectively, the problem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

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

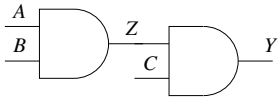
A satisfying assignment to Equ. 2 implies that $F_{function}$ can be realized in the programmable circuit.

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 [10]. This characteristic function describes all valid inputs, output, configuration bits, and internal signal vectors for the configurable circuit. For example, consider the truth-table in Fig. 4. Its onset describes all input-output relations of an AND gate. To extract the characteristic function, F_{AND}, from the truth-table, any standard minimization technique can be used.

A	B	Z	F_{AND}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



$$F_{\text{AND}} = (A + \overline{Z}) \cdot (B + \overline{Z}) \cdot (\overline{A} + \overline{B} + Z)$$



$$F_{\text{CASCADE}} = (A + \overline{Z}) \cdot (B + \overline{Z}) \cdot (\overline{A} + \overline{B} + Z) \cdot (Z + \overline{Y}) \cdot (C + \overline{Y}) \cdot (\overline{Z} + \overline{C} + Y)$$

Fig. 4. Deriving a circuit characteristic function.

Deriving characteristic functions directly from the circuit input-output relation is only practical for primitive gates and logic blocks where the number of inputs and outputs is small. Fortunately, characteristic functions for larger circuits can be derived iteratively from the conjunction of its subcircuit characteristic functions. For example, consider the cascaded gates shown in Fig. 4. Notice the wire connecting the two gates is labeled with variable Z for CNF construction. The characteristic function of the cascaded circuit is simply the conjunction of the two AND gate characteristic functions with variable Z as the logical link between the two functions. The characteristic function, F_{CASCADE} evaluates to true if all wire signals are consistent. This includes the primary inputs and outputs as in F_{AND}, plus any intermediate wire signals (i.e. Z).

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. 3 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 opera-

tion.

$$[G_{circuit} \equiv F_{function}] \equiv \Psi\,[G/F(x_1,...,x_n)] \\ \equiv \exists L_1...L_m \forall x_1...x_n \exists z_1...z_o \psi\,[G/F(x_1,...,x_n)] \quad (3)$$

4.2. Removing Quantified Variables

Although QSAT 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 [11]. 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 a 3-input function F needs to be implemented in the PLB shown previously in Fig. 3. In the following steps,F represents the function of the cone under consideration for mapping, $\boldsymbol{X_i}$ represents input vector $x_1x_2x_3 = i$, and $F_i = F(\boldsymbol{X_i})$.

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

$$G_{LUT} = (x_1 + x_2 + \overline{L_1} + z_1) \cdot (x_1 + x_2 + L_1 + \overline{z_1}) \cdot \\ (x_1 + \overline{x_2} + \overline{L_2} + z_1) \cdot (x_1 + \overline{x_2} + L_2 + \overline{z_1}) \cdot \\ (\overline{x_1} + x_2 + \overline{L_3} + z_1) \cdot (\overline{x_1} + x_2 + L_3 + \overline{z_1}) \cdot \\ (\overline{x_1} + \overline{x_2} + \overline{L_4} + z_1) \cdot (\overline{x_1} + \overline{x_2} + L_4 + \overline{z_1}) \quad (4)$$

$$G_{MUX} = (L_5 + \overline{x_3} + z_2) \cdot (L_5 + x_3 + \overline{z_2}) \cdot (\overline{L_5} + z_2) \quad (5)$$

$$G_{AND} = (z_1 + \overline{G}) \cdot (z_2 + \overline{G}) \cdot (\overline{z_1} + \overline{z_2} + G) \quad (6)$$

Step 2: Formulate the programmable circuit CNF from equations 4, 5, and 6.

$$G_{circuit} = G_{LUT} \cdot G_{MUX} \cdot G_{AND} \quad (7)$$

Step 3: Replication of equation 7 to remove quantified variables. This formulates G_{Total} where a satisfiable assignment to G_{Total} implies F can be realized in the programmable circuit.

$$G_{Total} = G_{circuit}[\boldsymbol{X}/\boldsymbol{X_0}, G/F_0, z_1/z_3, z_2/z_4] \cdot \\ G_{circuit}[\boldsymbol{X}/\boldsymbol{X_1}, G/F_1, z_1/z_5, z_2/z_6] \cdot \\ G_{circuit}[\boldsymbol{X}/\boldsymbol{X_2}, G/F_2, z_1/z_7, z_2/z_8] \cdot \\ G_{circuit}[\boldsymbol{X}/\boldsymbol{X_3}, G/F_3, z_1/z_9, z_2/z_{10}] \cdot \\ G_{circuit}[\boldsymbol{X}/\boldsymbol{X_4}, G/F_4, z_1/z_{11}, z_2/z_{12}] \cdot \\ G_{circuit}[\boldsymbol{X}/\boldsymbol{X_5}, G/F_5, z_1/z_{13}, z_2/z_{14}] \cdot \\ G_{circuit}[\boldsymbol{X}/\boldsymbol{X_6}, G/F_6, z_1/z_{15}, z_2/z_{16}] \cdot \\ G_{circuit}[\boldsymbol{X}/\boldsymbol{X_7}, G/F_7, z_1/z_{17}, z_2/z_{18}] \quad (8)$$

Note that in equation 8, 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.

In the previous example, the pins on the programmable circuit in Fig. 3 are not permutable. Given the labeling convention in Fig. 3, the function $F = (x_1 + x_2) \cdot x_3$ can be implemented; however, the function $F = (x_1 + x_3) \cdot x_2$ cannot. There is no need for restricting the labeling of the input pins in this manner because most programmable circuits are able to route signals to any input pins. In order to model this flexibility, virtual multiplexers controlled by virtual configuration bits, V_p, are added at each input pin of the programmable circuit. Going back to the circuit shown in the last example, Fig. 5 illustrates the previous circuit with virtual multiplexers added at the input pins. Thus, if $F = (x_1 + x_3) \cdot x_2$ is to be mapped into this network then the virtual multiplexers would force x_1 and x_3 onto the first two pins of the circuit and x_2 to the third pin feeding the AND gate to generate a satisfiable solution. In order to add the virtual multiplexers to the previous example, the virtual multiplexer characteristic functions need to be added in **Step 1**, then the the process proceeds normally as previously shown.

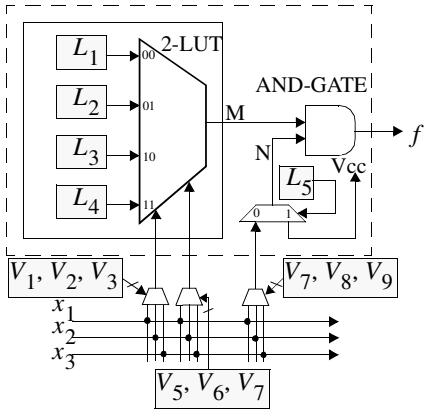


Fig. 5. An example PLB with virtual multiplexers added.

4.3. Application to PLB Evaluation

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

Fig. 6. An overview of the PLB evaluation algorithm.

Fig. 6 shows a high-level overview of our PLB evaluation algorithm. 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, our tool can be used to return a PLB cone fit percentage where a high fit percentage is preferred. This fit percentage is found by taking extracting a set of cones from a list of circuits (Fig. 6, line 1), then applying our QSAT decision step to remove cones that do not fit in the given architecture (Fig. 6, line 2). By recording the number of cones generated and discared, a fit percentage for various PLB architectures can be found (Fig. 6, line 3).

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 primary inputs to primary outputs. At every internal node v, new cones are generated by combining the cones at the input nodes.

5. RESULTS

5.1. Evaluation of Various PLBs

To show the power of the PLB evaluation algorithm, several unrelated PLB architectures were evaluated. Fig. 7 shows the five different PLB architectures used for evaluation.

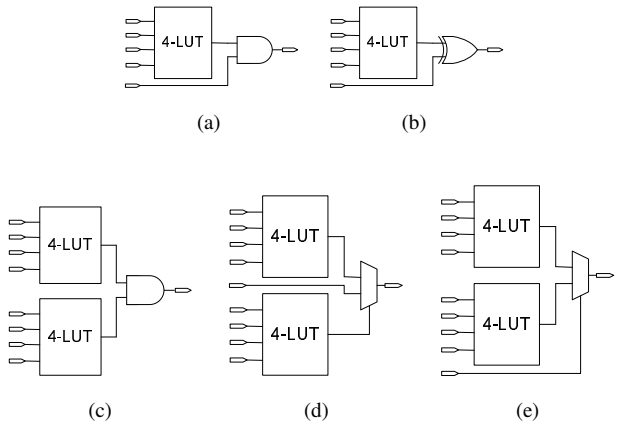


Fig. 7. PLB architectures.

To evaluate the versatility of each PLB, a set of cones were extracted from a list of circuits taken from the MCNC benchmark suite [12] (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 [13] SAT solver. The circuits used were unrelated to generate a large set of dissimilar cones. The table shown in Tab. 1 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 very rare in most control circuits and are generally used for arithmetic logic.

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 1. PLB fit results.

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

Table 2. SAT and QBF solver running times.

5.1.1. QBF vs. SAT

In order to show the power of removing quantifiers in our QBF to produce a SAT problem as shown in Sec. 4.2, we ran a few cone fitting examples using a QBF solver and a SAT solver. The example used a 7-input function realized with a 7-input PLB consisting of two cascaded 4-LUTs. An unsatisfiable function was selected so the entire search space was explored. Furthermore, the configuration bits were preconfigured to vary the size of the search space. This is shown in Tab. 2. *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 [14] running times on the same machine.

6. CONCLUSION AND FUTURE WORK

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 the PLB evaluation tool. QSAT is P-Space 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.

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.

7. REFERENCES

[1] Altera, "Component selector guide ver 14.0," 2004.

[2] Xilinx, "Virtex-ii complete data sheet ver 3.3," 2004.

[3] Arrow Electronics. [Online]. Available: www.arrow.com

[4] 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, pp. 1–13, Jan. 1994.

[5] ——, "On area/depth trade-off in LUT-based FPGA technology mapping," in *Design Automation Conference*, 1993, pp. 213–218. [Online]. Available: citeseer.ist.psu.edu/cong94areadepth.html

[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*. ACM Press, 1999, pp. 29–35.

[7] I. Levin and R. Y. Pinter, "Realizing Expression Graphs using Table-Lookup FPGAs," in *Proceedings of the European Design Automation Conference*, 1993, pp. 306–311.

[8] A.Kaviani and S. Brown, "The hybrid field programmable architecture," *IEEE Design and Test*, pp. 74–83, April–June 1999.

[9] 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*, vol. 20, no. 9, pp. 1077–1090, 2001.

[10] T. Larrabee, "Test Pattern Generation Using Boolean Satisfiablity," *IEEE Transactions on Computer-Aided Design*, vol. 11, no. 1, pp. 6–22, 1992. [Online]. Available: citeseer.ist.psu.edu/larrabee92test.html

[11] 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*, May 2004, pp. 10–13.

[12] S. Yang, "Logic synthesis and optimization benchmarks user guide version," 1991. [Online]. Available: citeseer.ist.psu.edu/yang91logic.html

[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. [Online]. Available: citeseer.ist.psu.edu/moskewicz01chaff.html

[14] 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*. ACM Press, 2002, pp. 442–449.