Technology Mapping for a Two-Output RAM-Based Field Programmable Gate Array

David Filo *Jerry Chih-Yuan Yang* *Frédéric Mailhot* *Giovanni De Micheli*

Center for Integrated Systems
Stanford University
Stanford, CA 94305

Abstract

We present a new approach for performing *technology mapping* onto *Field Programmable Gate Arrays* (FPGAs). We consider one class of FPGAs, based on two-output five-input RAM-based cells, that are used to implement combinational logic functions. We describe a heuristic algorithm for technology mapping that performs a decomposition of the circuit in the FPGA primitives, driven by the information on logic functional sharing. We have implemented the algorithm in the program *Hydra*. Experimental results shows an average of 20 % to 25 % improvement over other existing programs in mapping area and 67-fold speedup in computing time.

1 Introduction

There has been an increasing interest in digital-system prototyping using *Field Programmable Gate Arrays* (FPGAs) due to their fast turn-around time and low manufacturing costs. One class of FPGAs uses a RAM-based architecture, where logic blocks in the form of look-up tables are used to implement combinational logic. The advantage of this architecture is that a logic block can perform any combinational function of its inputs.

System design with FPGAs requires specific logic design tools. In particular, *technology mapping* is crucial for achieving an efficient implementation. Technology mapping is the process of transforming a set of logic equations into an interconnection of parts that are instances of the elements in a given library. In the case of FPGAs, the "library" consists of the set of combinational logic equations satisfying constraints on the number of I/Os and their dependencies. Since this set is large and since it can be expressed more conveniently by the constraints than by enumeration, no specific library representation is used.

Existing approaches to technology mapping include algorithms and tools that support an explicit library definition, such as *MisII* [3] and *Ceres* [7]. These tools are inefficient for FPGAs because they require an explicit library representation. Technology mapping algorithms for cell generators [2] were developed to be used in conjunction with module generators that could synthesize combinational logic gates under some technology constraints. They can be used for single-output FPGAs, but they cannot exploit the multiple-output feature of some FPGAs.

Recently, several specific mappers for RAM-based FPGAs have been developed in addition to the proprietary Xilinx *XNFOPT* [10] mapper. Mapping of FPGAs with emphasis on placement and routing is described in [1] and is implemented in the ASYL system. Other mappers which target the reduction of area include *Chortle* [5], which uses a dynamic programming approach for mapping table look-up architectures. The *Chortle* approach was shown to be consistently better than the one in *MisII* for look-up tables with fan-in of 3, where a "library" could be generated exhaustively. This algorithm is limited to single-output lookup tables; however, the approach has been recently extended in *Chortle-crf* [9]. The algorithm of *Chortle-crf* performs technology mapping for single-output functions in a first phase, and later it attempts to merge functions into multi-output blocks. Another mapper for RAM-based architecture is *Mis-PGA* [8]. This algorithm also focuses on single-output cells first, and then considers merging.

This paper describes a new two-output RAM-based technology mapper called *Hydra*, which maps combinational logic networks to FPGAs, such as the Xilinx 3000 Gate Array Architecture [11]. This approach differs from the others mentioned in that multiple-outputs are considered early in the mapping process. Experimental results show that this choice correlates to superior results. Presently, *Hydra* is constrained to combinational circuits, and is aimed to minimize mapping area. The target architecture being considered consists of an array of *Configurable Logic Blocks* (CLBs). At present, the largest gate array in the Xilinx 3000 family supports 320 CLBs. A CLB is a RAM-based cell that can implement one of the following:

- Any single-output logic function of up to five input variables.
- Any two-output logic function of up to five input variables, with each output depending on at most four input variables.

The technology mapping problem can be divided into two phases:

1. *Feasible mapping*. Map a combinational logic network into an interconnection of CLBs satisfying the above constraints.
2. *Optimal mapping*. Minimize the number of CLBs in a feasible mapping.

Effectiveness of the second phase is heavily influenced by the actions of the first phase. The first phase attempts to produce expression pairs which can later be merged into one CLB.

The rest of the paper is organized as follows. A detailed description of the mapping algorithms is given in Section 2. In Section 3, experimental results are presented and compared to those obtained by *Mis-PGA* and *ASYL*. Extensions and future directions are then presented.

2130/91/0000/0534$01.00 © 1991 IEEE

2 The Mapping Algorithm

Combinational circuits are modeled by a directed acyclic graph called a *Boolean network* [3]. *Vertices* in the Boolean network represent logic functions, and *edges* represent the dependencies among them. The direction of the edges of the network is from primary inputs to primary outputs. Given vertices v_i and v_j, v_i is a *predecessor* (*successor*) of v_j if there exists a directed edge from (to) v_i to (from) v_j.

For each vertex v_i in the Boolean network there is an associated variable and a function $f_i(S_i)$, where S_i is the support of f_i (i.e. S_i is the set of variables corresponding to the predecessors of v_i).

Technology mapping for FPGAs is based on repetitive decomposition of logic functions to achieve feasibility. In general, a decomposition of a function f with support S can be written as follows:

$f = g(h(S^A), S^B)$ where $S^A, S^B \subseteq S$ and $S^A \cup S^B = S$

If in addition, $S^A \cap S^B = \emptyset$, then g is a *simple-disjoint* decomposition of f [4] [6].

A simple-disjoint decomposition is attractive because it allows a function to be decomposed very efficiently. When the function f is decomposed into functions h and g as shown above, the combined support for the two functions has cardinality $|S^A| + |S^B| + 1$. This is a minimum when $S^A \cap S^B = \emptyset$ (i.e. the decomposition is disjoint).

The guiding principle in the technology mapping algorithm presented here is to use the disjoint decompositions in conjunction with a method that tries to exploit the use of both CLB outputs. Therefore special attention is paid to function pairs having common support variables.

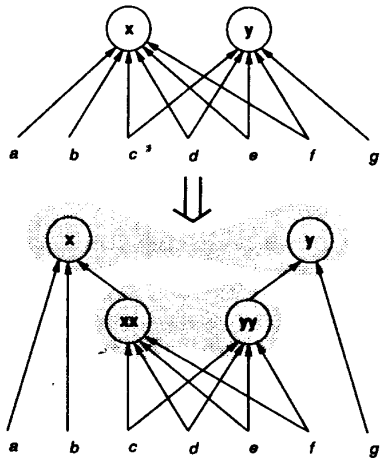

Figure 1: Example of CLB Mapping Using Shared Inputs

Consider the example network shown in Figure 1. The vertices x and y have a combined total of 7 predecessors, 4 of which are common to both. If f_x and f_y were decomposed independently, then at least 3 CLBs would normally be required because the support of f_x is greater than five. However, by considering the common predecessors, it is possible to map the network into 2 CLBs, provided an appropriate disjoint decomposition can be found for both f_x and f_y as shown in the figure.

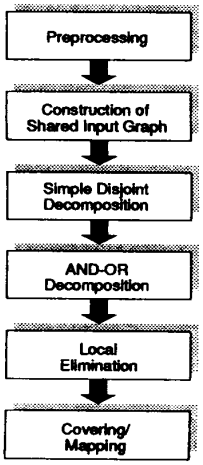

Figure 2: Block Diagram of *Hydra's* Mapping Steps

The mapping algorithm can be summarized as the following two tasks. First, a feasible mapping is found by using two decomposition techniques, the *simple-disjoint* and the *AND-OR* decompositions. Simple-disjoint decompositions are the most desirable. They are applied to function pairs to extract common predecessors, as described in detail later. Unfortunately, simple-disjoint decompositions may not be sufficient for generating a feasible network. Repetitive application of the AND-OR decomposition yields feasible networks, but it does not exploit logic functional sharing. Since the latter decomposition technique is much faster than the former, it is applied twice. In the preprocessing stage, the AND-OR decomposition splits functions with large support to make the simple-disjoint decomposition more efficient. AND-OR decomposition is applied again after the simple-disjoint decomposition to insure feasibility of the network.

The second task is the search for an optimal mapping. A greedy heuristic algorithm is used to match the vertices of the feasible network to CLBs. A cost function based on the CLB's number of shared inputs and input utilization is used to determine the best merging candidate for a given vertex. A block diagram of the mapping steps is presented in Figure 2.

We now describe the steps of technology mapping in more detail.

2.1 Preprocessing

We assume that logic synthesis and optimization algorithms have been applied to the network so that it is minimal in the number of literals.

An initial decomposition is performed to decompose functions at vertices with in-degree larger than a given threshold. The input is a Boolean network in AND/OR form. The result of this phase is an equivalent Boolean network in which the in-degree of vertices are bounded by the input limit specified by the user. The AND-OR decomposition, described in Section 2.4, is used for this step.

2.2 Construction of Shared-input Graph

It has been observed that in a Boolean network a large number of vertices have common predecessors. This is particularly true of networks that have been optimized using kerneling techniques [3]. These shared-input relationships are used to drive the simple-disjoint de-

composition. This information is represented by a weighted graph $G(V, E, W)$. The vertex set V corresponds to the one in the Boolean network. For each pair of vertices $v_i, v_j \in V$ that have common predecessors, there is a corresponding edge $e_{ij} \in E$ with a weight $w_{ij} \in W$, where w_{ij} is the number of common predecessors. This graph is constructed by traversing the Boolean network using an algorithm with complexity $O(n^3)$, where n is number of vertices.

2.3 Simple-Disjoint Decomposition

Each edge $e_{ij} \in E$ of the shared-input graph is traversed to check for simple-disjoint decompositions into functions with support common to both v_i and v_j. The edges with the highest weight are traversed first so that decompositions with a high degree of sharing are found. Once a disjoint decomposition is found, a cost function is used to decide whether or not to accept it. The cost is based on the number of shared inputs used as well as the overall number of inputs in the extracted functions.

The algorithm is based on the notion of *residues* presented in [6]. Assume that we search for a disjoint decomposition

$$f(S) = g(h(S^A), S^B) \text{ with } S^A \cap S^B \equiv \emptyset$$

Let R^B be the set of residues of f obtained by replacing the variables in S^B by constants. The function f has a simple-disjoint decomposition g if and only if $(\forall r \in R^B)\ r \in \{0, 1, h(S^A), h'(S^A)\}$ [6]. By testing each residue r, it can be determined whether or not g is a valid decomposition of f.

Checking a residue is exponential in $|S^B|$. Since the number of different sets S^A is factorial in $n = |S|$, the overall complexity of checking for every possible disjoint decomposition of f has complexity $O(n!2^n)$. It is important to remark that n is small because of the preprocessing step, which limits the fan-in of the vertices. Furthermore, far less than all possible combinations need to be checked, because only pairs of decompositions having shared support are considered.

2.4 AND-OR Decomposition

After the simple-disjoint decomposition is applied to the network, the AND-OR decomposition is used to decompose the remaining infeasible vertices. Functions are represented by Boolean factored forms (including *sum of product* representations) [3], and can be described by trees, as shown in Figure 4.

A heuristic approach that performs a post-order traversal of the equation tree is used. At each vertex of the tree the algorithm described in Figure 3 is applied.

```
while (support of predecessors > limit) do
    find group of predecessors with combined support ≤ limit
    combine these into a new vertex in the Boolean network
    replace predecessors with the new vertex
end while
```

Figure 3: AND-OR Decomposition Heuristic

It is always possible to find a combination of predecessors that have a combined support of no more than $limit$. This is insured by use of a post-order traversal of the equation tree.

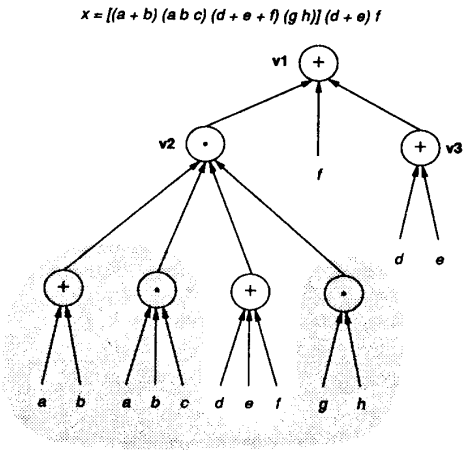

Figure 4: Equation Tree for a Vertex of the Boolean Network

Heuristics are used to select the best combination of predecessors to combine. The following parameters are used to characterize a particular combination.

- The size of overall support
- The maximum support of a single vertex
- The number of vertices in the combination

For example, the best combination (for $limit = 5$) of predecessors for the vertex v_2 in Figure 4 is indicated by the shading. The corresponding function (i.e. $(a + b)(abc)(gh)$) is extracted and associated to a new vertex of the Boolean network. The complexity of this algorithm for each vertex is $O(n^2)$, where n is the cardinality of the support of the function.

2.5 Local Elimination

At this point, a local elimination [3] of vertices is performed. Starting from the primary inputs of the network, vertices with one or more successors are tested to see if they can be eliminated into their successors. If the resulting vertices are still feasible, then the elimination is performed. When a vertex has only one successor, elimination is allowed if the in-degree of the resulting vertex satisfies an upper bound of 5. When a vertex has multiple successors, the bound is lowered to 4 to allow pairs of successor vertices to share a CLB. This step reduces the number of vertices in the final implementation and provides better utilization of CLB inputs.

2.6 Covering/Mapping

The covering step is performed last. Each vertex is compared with every other vertex in the network. If two vertices can be merged into one CLB, then a cost is calculated based on the following formula.

$$COST = \alpha(shared\_inputs) + \beta(total\_inputs)$$

Different weights can be chosen to emphasize different mapping preferences. For example, if one wants to map CLBs based on higher shared-input usage, then a higher weight can be given to α. Similarly,

if better utilization of CLB inputs is desired, then β can be given a higher weight. Better results have been obtained by placing emphasis on shared inputs.

Vertices with in-degree equal to 5 are mapped first, because they require one CLB. Then the following steps are iterated:

1. An unassigned vertex with maximal in-degree is selected;
2. A second unassigned vertex is chosen, so that the pair has the highest cost;
3. The pair is assigned to a CLB.

Eventually the vertices that cannot be paired to others are mapped to a CLB. The complexity for the covering heuristic is $O(n^2)$, where n is number of vertices in the Boolean network.

Circuit	Hydra	Mis-PGA	Difference	Time
10bitreg	10	10	0	0.2
10count	22	23	-1	0.5
180degc	22	21	1	0.5
3to8dmux	20	30	-10	0.6
4-16dec	10	12	-2	0.2
4cnt	12	17	-5	0.9
8bappreg	23	27	-4	0.6
8count	29	20	9	1.0
9bcsac1	28	34	-6	0.8
9bcasc2	27	29	-2	0.7
arbiter	22	21	1	0.6
5xp1	21	23	-2	0.7
C499	51	50	1	1.9
C5315	302	497	-195	15.4
apex6	159	191	-32	7.7
apex7	45	50	-5	1.0
duke2	80	105	-25	3.9
rd84	29	32	-3	1.1
rot	134	153	-19	6.4
vg2	21	21	0	0.5
Total	**1067**	**1366**	**-299**	**45.2**

Table 1: Mapping Results (Number of CLBs), input limit = 9

3 Implementation and Results

Hydra consists of about 2500 lines of C code, based on the Structural/Logic Intermediate Form (SLIF) framework [7]. Two sets of benchmarks have been run. The first set is a series of industrial examples. The second is a set of selected files from the standard MCNC benchmarks. All files have been logically optimized using the *MisII* standard script beforehand.

3.1 Analysis

In the first set, *Hydra* uses an input limit (i.e. the threshold for preprocessing) of 9, $\alpha = 3$ and $\beta = 1$. Results are compared against those obtained from *Mis-PGA* [8] in Table 1. The run times (on a DECstation 3100) listed in Table 1 range from less than 1 second for most benchmarks to at most 16 seconds on the largest one. The total run time of 45.2 seconds is compared to *Mis-PGA's* total run time of 8163 seconds (On a VAX-8800). Taking the hardware performance ratio [1] into account, *Hydra* runs approximately 68 times faster. For the benchmarks in Table 1 *Hydra* results in 22% fewer CLBs than *Mis-PGA*. It should be noted, however, the *C5315* benchmark places a large bias on the overall result.

By specifying an *input limit* in the preprocessing step, we control the granularity of the network. For circuits with highly connected shared-input graphs (those with more sharing), such as *C5315*, the input limit is used to reduce the computation time of finding disjoint decompositions. In general the input limit for an optimal mapping is different for every network. Although the results in Table 1 are not the best mapping possible, they are not far from optimal. By varying the input limit, it is possible to achieve an additional overall reduction of 45 CLBs for the benchmarks in Table 1.

In the second set of results, additional MCNC benchmark circuits are mapped. The best possible mappings are obtained by varying the input limit. They are listed in Table 2. These results are compared with *Mis-PGA* and *ASYL*, when their results are available. It is observed that a lower input limit gives better results for small to medium sized circuits. *Hydra* consistently gives better results than *Mis-PGA* in larger circuits, while both mappers give comparable results for smaller circuits. For the benchmarks in Table 2, *Hydra* results in 26% fewer CLBs than *Mis-PGA*. When compared to other existing mappers on the same set of benchmarks, *Hydra* is 30% better than *XNFOPT* and comparable in quality to *Chortle-crf* [9]. Selected benchmark results have been verified by the logic simulator in *Mercury* [7].

Name	Hydra		Mis-PGA		ASYL
	CLBs	Time[a]	CLBs	Time[b]	CLBs
5xp1	21	0.5	23	45.5	23
9sym	57	2.9	59	-	-
9symml	33	1.1	43	-	-
C499	51	1.8	50	137.5	-
C5315	299	23.0	497	3467	-
C880	71	9.0	82	-	-
alu2	94	2.9	102	-	-
alu4	105	4.8	189	-	-
apex2	67	3.5	70	-	-
apex6	131	35.9	191	1377	-
apex7	43	0.9	50	117.3	-
count	26	0.5	28	-	-
duke2	79	1.9	105	357.1	-
e64	47	0.7	61	-	-
misex1	8	0.2	10	-	14
rd84	27	0.6	32	-	-
rot	134	6.4	153	844.8	-
vg2	20	0.3	21	25.6	21
z4ml	4	0.1	7	-	-
rd73	13	0.4	-	-	23
misex2	20	0.4	-	-	24
sao2	36	1.2	-	-	38

[a] DECstation 3100
[b] VAX-8800

Table 2: MCNC Benchmark Results

3.2 Comparison With *Mis-PGA* Approach

The main difference between the *Hydra* algorithm and FPGA mappers that focus on mapping one-output cells (such as *Mis-PGA*) is the way in which multiple outputs are treated. For example, in *Mis-PGA*, merging of functions to form multi-output CLBs is performed as a last step. Murgai [8] reports that *merge* improves *Mis-PGA* results

[1] The DECstation runs 24000 Dhrystones per second, while the VAX-8800 runs 9000 per second. This translates to a ratio of 2.67.

Circuit	% Multi-Output	% Shared Inputs
10bitreg	100.0	0.0
10count	54.2	29.2
180degc	59.1	40.9
3to8dmux	68.0	76.2
4-16dec	100.0	100.0
4cnt	83.3	58.3
8bappreg	68.0	48.0
8count	81.5	59.3
9bcasc2	73.1	61.5
9bcsac1	74.1	51.8
arbiter	73.9	60.9
5xp1	54.5	68.2
C499	30.4	48.2
C5315	48.6	33.2
apex6	64.4	56.2
apex7	43.5	54.3
duke2	61.0	54.9
rd84	48.4	41.2
rot	61.8	48.5
vg2	34.8	34.8

Table 3: Percentage of Cells with Multi-outputs and Shared Inputs

by 5-15%. This means that the majority of the CLBs mapped by *Mis-PGA* contain one output only. In *Hydra*, the mapping algorithm is aimed at merging multiple outputs into one CLB. From Table 3, an average of 64% of the CLBs mapped by *Hydra* contain multiple outputs.

Like *Mis-PGA*, *Hydra* makes an effort to minimize routing complexity, without performing a routability analysis. One measure of wiring costs is the number of edges created as a result of a decomposition [8]. In *Hydra*, disjoint decompositions yield a minimum number of edges; hence routing costs are reduced. Targeting shared inputs also eliminates many of the wiring costs, since a shared signal is used in two logic functions. The percentage of CLBs implementing two functions with some shared inputs is reported in Table 3. An average of 51% of CLBs fall in this class, indicating the efficiency of targeting shared inputs.

4 Conclusions and Future Work

This work has shown a new approach to technology mapping for two-output RAM-based FPGAs. The main idea in our approach is to use the shared-input relationship in a network to drive disjoint decompositions. Results have shown that, compared to other tools, this approach is competitive for small circuits, and consistently superior in larger ones. In particular, results show an improvement of about 20% in the number of cells and a 60-fold speed-up in computing time with respect to *Mis-PGA*. *Hydra* can be easily extended to map any m input, 2-output RAM-based architecture.

Future work will address expanding the search for other types of disjoint decompositions; for example, to include the *don't care* sets. It is also interesting to evaluate the efficiency of other possible architectures by extending to RAM-based FPGAs with more than two outputs.

5 Acknowledgment

This research was sponsored by NSF-ARPA, under grant No. MIP 8719546 and, by AT&T and DEC jointly with NSF, under a PYI Award program. We acknowledge also support from ARPA, under contract No. J-FBI-89-101.

References

[1] P. Abouzeid, L. Bouchet, K. Sakouti, and P. Sicard G. Saucier. Lexicographical expression of boolean functions for multilevel synthesis of high speed circuits. In *Proceedings of the Synthesis and Simulation Meeting and International Interchange*, pages 31–39, Kyoto, October 1990.

[2] M. R. C. M. Berkelaar and J. A. G. Jess. Technology mapping for standard-cell generators. In *Proceedings of the International Conference on Computer-Aided Design*, pages 470–473, Santa Clara, November 1988.

[3] R.K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A.R. Wang. MIS: A multiple-level logic optimization system. *IEEE Transactions on CAD/ICAS*, pages 1062–1081, November 1987.

[4] H.A. Curtis. *A New Approach to the Design of Switching Circuits*. D. Van Nostrand Company, Princeton, N.J., 1962.

[5] R.J. Francis, J. Rose, and K. Chung. Chortle: A technology mapping program for lookup table-based field programmable gate arrays. In *Proceedings of the 27th Design Automation Conference*, pages 613–619, Orlando, June 1990.

[6] E.J. McCluskey. *Logic Design Principles*, pages 182–185. Prentice Hall, 1986.

[7] G. De Micheli, D. Ku, F. Mailhot, and T.K. Truong. The Olympus synthesis system for digital design. *IEEE Design and Test*, pages 37–53, October 1990. and Stanford Technical Report, CSL-TR90-432.

[8] R. Murgai, Y. Nishizaki, N. Shenoy, R.K. Brayton, and A. Sangiovanni-Vincentelli. Logic synthesis for progammable gate arrays. In *Proceedings of the 27th Design Automation Conference*, pages 620–625. Orlando, June 1990.

[9] J. Rose. Private communication, 1990.

[10] Xilinx Inc. *XACT LCA Development System, Vol. II*, 1989.

[11] Xilinx Inc. *Xilinx Programmable Gate Array User's Guide*, 1989.