Technology Mapping for Field-Programmable Gate Arrays Using Integer Programming[1]

Amit Chowdhary and John P. Hayes
Advanced Computer Architecture Laboratory
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109-2122
{*amitc,jhayes*}*@eecs.umich.edu*

Abstract

We show that the FPGA technology mapping problem can be efficiently implemented as a mixed integer linear programming (MILP) problem which generates truly optimal mappings. The MILP approach can handle a wide variety of FPGA logic block architectures. We present a compact MILP formulation for logic blocks based on lookup tables (LUTs) or multiplexers. We also show that the MILP formulation can be easily modified to optimize area, delay, or a combination of both. We demonstrate that moderately large benchmark circuits can be mapped in a reasonable time using the MILP approach directly. For larger circuits, we propose a technique of partitioning a circuit prior to mapping, which drastically reduces the computation time with little or no loss in optimality.

1 Introduction

Field-programmable gate arrays (FPGAs) are user-programmable devices intended for rapid prototyping and small-scale production of digital circuits. An FPGA is a two-dimensional array of logic blocks which can be interconnected by programmable routing wires and switches placed in the channels between the rows and columns of logic blocks. The logic blocks can implement a large variety of functions of a few input variables—up to 5 or so. Current FPGAs can be classified into two types based on their logic block structure: lookup table-based and multiplexer-based. An m-input lookup table (LUT) is a static RAM which can implement any function of at most m inputs. A multiplexer-based logic block consists of a multi-level arrangement of small multiplexers (muxes). Figure 1 shows the logic blocks used in various FPGAs [1, 16].

The FPGA technology mapping problem is to map a logic circuit onto the logic blocks of an FPGA, while minimizing an objective function such as area, delay, or a combination of both. FPGA mapping cannot be performed efficiently using conventional library-based techniques due to the large number of functions which can be implemented by a logic block. We present an exact approach to FPGA mapping, where the problem is efficiently formulated as a mixed integer linear programming (MILP) problem. The general MILP problem [14] is to minimize a linear objective function in terms of integer or real variables subject to some linear constraints.

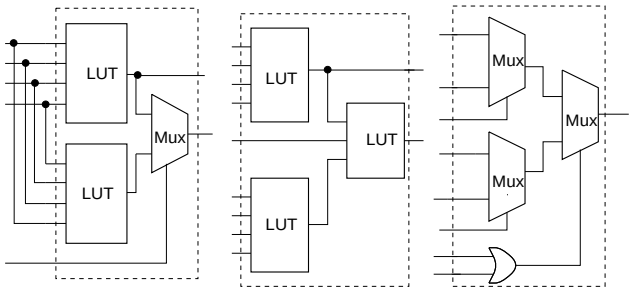

Figure 1: FPGA logic blocks: (a) the LUT-based Xilinx $X3000$[16]; (b) the LUT-based Xilinx $X4000$[16]; and (c) the mux-based Actel $ACT1$[1].

The proposed MILP-based approach to FPGA mapping has the following advantages over the existing techniques which employ heuristics.

1. It generates optimal mappings, which are often much better than those produced by conventional heuristic techniques, as demonstrated by our experimental results. For example, our technique gives an optimal mapping of 64 5-input $X3000$-style LUTs for the ISCAS-85 benchmark circuit c499 [2]. The best result reported for c499 using other techniques [9] is 74 5-input LUTs.
2. It is a very flexible approach, since a variety of logic blocks composed of LUTs, muxes, or indeed, any logic structure, can be accommodated. Existing techniques are restricted to one particular

[1] This research was supported by the National Science Foundation under Grant No. MIP–9200526.

logic block structure.

3. It can be easily modified to optimize area, delay or a combination of both, just by changing the objective function or adding a few constraints. Prior techniques are designed to optimize just one of these goals.
4. Moderately large circuits can be mapped optimally in a small amount of time using the MILP approach directly.
5. Very large circuits can be mapped near-optimally by partitioning the circuits and mapping each partition individually using the MILP approach.

The paper is organized as follows. Section 2 defines the mapping problem. Section 3 briefly reviews the general technique for solving MILP problems. Section 4 describes our MILP formulation for LUT-based FPGAs. Section 5 explains the circuit partitioning approach to speed up the algorithm. Section 6 addresses the mux-based logic blocks. The experimental results are presented in Section 7.

2 Problem Statement

The input to the FPGA technology mapping process is a gate-level description of a combinational logic circuit C. We model C by a directed graph G, where the gates are represented by nodes, and the interconnections are represented by edges. We may also use functional information about C, such as the type and inversion parity of the gates, for example when using multiplexer-based logic blocks. The type and inversion parity of gates are ignored in the case of LUT-based logic blocks, since a LUT of size m can implement any function with at most m inputs. Figure 2 shows a small example. The technology mapping process is often preceded by a technology-independent logic minimization step performed using, for example, the *mis* logic synthesis package [6]. In our experiments, we perform the same technology-independent minimization to make our results comparable with prior work.

We make the following general assumptions about the circuit C.

1. Every gate in C has a fanin less than or equal to the size m of the logic block.
2. Inverters are merged with the input lines of the gates. This reduces the size of C and G, which improves the execution speed of MILP approach.
3. When a node v fans out to multiple nodes, we can replicate a subcircuit rooted at v for every outgoing edge. This subcircuit replication can reduce the number of LUTs, as explained below.

In general, the *FPGA technology mapping problem* is to map the graph G using the logic blocks of a given FPGA F such that:

1. Every node of G is included in at least one logic block of F.

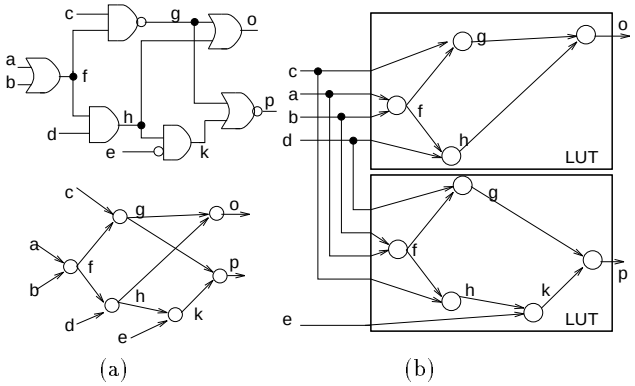

Figure 2: Mapping using 5-input LUTs: (a) circuit C and its graph G, and (b) optimal cover using two LUTs.

2. The interconnections among the logic blocks of F follow those of G.
3. The constraints defining the logic blocks of F are met. In the case of logic blocks with a single LUT of size m, the numbers of inputs and outputs are bounded by m and 1 respectively.
4. An objective, such as area, delay or a combination of both, is optimized.

The set of logic blocks selected to map a circuit is called a *cover* of G. The circuit of Fig. 2 can be optimally covered by two 5-input LUTs, such as those in the Xilinx $X3000$ logic blocks, as shown in Fig. 1. In this example, nodes f, g and h are replicated to reduce the number of LUTs.

The mapping problem for logic blocks can be solved to optimize various objective functions. The area and delay objectives refer to the number of logic blocks and the number of levels of logic blocks respectively, in the cover of G. An extensive survey of heuristic mapping algorithms for various logic block structures and mapping objectives is presented in [15]. A representative technique using LUTs for area minimization is Chortle-crf [9], which considers various decompositions of a gate using an approximate bin-packing technique, but exploits node replication only to a limited extent. The mapping problem under area minimization has been shown to be NP-complete for LUTs of size greater than 4 [7]. On the other hand, an optimal-delay cover of a circuit can be generated using a polynomial-time algorithm called FlowMap [4], but the area of the cover is unbounded. A modification of FlowMap [3] determines the optimal-delay mapping, and then allows a fixed increase in delay to reduce area. However, a more practical objective would be to reduce delay with the area fixed, since an FPGA contains a fixed number of LUTs. As explained later, our MILP formulation can optimize either area or delay while restricting the other parameter.

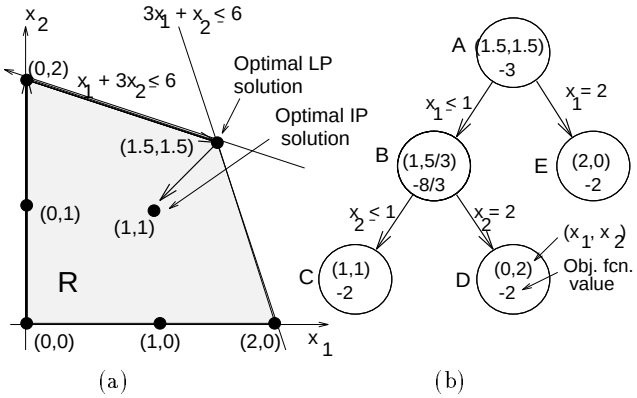

Figure 3: Solving an MILP problem: (a) the feasible region R (shaded); (b) the branch-and-bound tree.

3 Mixed Integer Linear Programming

The goal of mixed integer linear programming (MILP) is to minimize (or maximize) a linear objective function of a set of integer or real variables while satisfying a system of linear constraints [14]. The MILP problem is stated below in the usual matrix notation.

$$\begin{array}{ll} \text{Minimize} & \mathbf{cx} + \mathbf{dy} \\ \text{subject to} & \mathbf{Ax} + \mathbf{Dy} \leq \mathbf{b}, \\ & \mathbf{x} \geq \mathbf{0},\ \mathbf{y} \geq \mathbf{0} \end{array}$$

Here $\mathbf{c}$ and $\mathbf{d}$ are cost vectors of size n_1 and n_2, respectively; $\mathbf{A}$ and $\mathbf{D}$ are constraint matrices of size $m \times n_1$ and $m \times n_2$, respectively; $\mathbf{b}$ is a column vector of size m; $\mathbf{x}$ is a column vector of n_1 integer variables; and $\mathbf{y}$ is a column vector of n_2 real variables. The integer variables $\mathbf{x}$ usually have known upper bounds. If the MILP problem has no integer variables, *i.e.* $n_1 = 0$, then it reduces to a linear programming (LP) problem. If there are no real variables, *i.e.* $n_2 = 0$, then we have an integer programming (IP) problem.

The linear programming problem can be solved efficiently by the classic simplex algorithm [5]. We outline this algorithm using the following small example.

$$\begin{array}{ll} \text{Minimize} & -x_1 - x_2 \\ \text{subject to} & x_1 + 3x_2 \leq 6, \\ & 3x_1 + x_2 \leq 6, \\ & 0 \leq x_1 \leq 2, 0 \leq x_2 \leq 2 \end{array}$$

This example is represented graphically in Fig. 3. All possible solutions of the above system of inequalities are enclosed in the shaded region R, which is called the feasible region. The simplex algorithm searches for an optimal feasible solution on the boundary of R. It starts with a feasible solution corresponding to a corner of R. The algorithm moves from one corner to an adjacent corner until a corner corresponding to an optimal LP solution is reached. For the current example, the path traced by the algorithm is from $(0,0)$ to $(0,2)$ and then to $(1.5,1.5)$, which is the optimal solution.

We now briefly explain a widely-used method for solving an MILP problem which employs a branch-and-bound technique for implicitly enumerating the values of integer variables $\mathbf{x}$. At every step in the algorithm, we solve a subproblem of the original MILP problem where a subset of integer variables are fixed and the remaining variables are considered to be real. This is called an LP relaxation subproblem. Only a small portion of the tree is explored if efficient pruning criteria are used. Consider the above LP example where now x_1 and x_2 are required to be integers. The feasible MILP solutions are shown in Fig. 3(a). The LP relaxation problem has an optimal solution of $(x_1, x_2) = (1.5, 1.5)$ represented by node A of Fig. 3(b). We branch on variable x_1, and solve the subproblem corresponding to $x_1 \leq 1$. The next optimal solution sets x_2 to $\frac{5}{3}$ (node B). A feasible IP solution is found at the third node C where $x_2 \leq 1$. The optimality of the IP solution corresponding to node C is verified after nodes D and E are traversed.

4 LUT-based FPGAs

We first present the MILP formulation for technology mapping of LUT-based logic blocks to minimize various objectives.

Area minimization: When graph G is covered by a set of LUTs, a node of G can be mapped on the output of a LUT, or it can be embedded inside a LUT. We term the node external in the former case, and internal otherwise. In Fig. 2, the only external nodes are o and p. Our MILP formulation assigns a binary decision variable *external* to every node of G: *external*$[i]$ is 1 if node i is external, and 0 if node i is internal. The objective is to minimize the number of LUTs, which is just the sum of *external* variables over all nodes.

We have to ensure that the size of every LUT is less than or equal to m. We therefore define a *size* variable for every node, where *size*$[i]$ is the number of LUTs (or external nodes) which feed the LUT containing node i. In the case of circuits with no reconvergent fanout, *size*$[i]$ can be calculated as the sum of *size* variables for all its fanin nodes. If fanout from node s reconverges at node t, then *size*$[s]$ is counted multiple times while calculating *size*$[t]$. In order to avoid multiple counting, we introduce a variable *reconvergence* for every such node-pair $[s,t]$ which is made equal to *size*$[s]$, and subtracted while evaluating *size*$[t]$. The graph in Fig. 2 has two reconvergent node-pairs, $[f,o]$ and $[f,p]$.

Our MILP formulation has various types of constraints involving the above variables. We explain these constraints below using the example in Fig. 2.

1. *Boundary conditions*: The *external* variables are set to 1 for primary inputs and outputs.
2. *Size constraints*: The *size* variable for every node should be less than or equal to the LUT size m. In Fig. 2, *size*$[o] = 4$ and *size*$[p] = 5$, where $m = 5$.

3. *Reconvergence evaluation*: If the source s and sink t of a reconvergent node-pair are mapped into the same LUT, then the variable $reconvergence[s,t]$ is set to $size[s]$, since $size[s]$ is counted twice for $size[t]$. In the LUT cover in Fig. 2, the *reconvergence* values for $[f,o]$ and $[f,p]$ are both set to $size[f] = 2$. The following constraint ensures that $reconvergence[s,t] = size[s]$ in the optimal solution.

$$reconvergence[s,t] \leq size[s]$$

If the source s and sink t are in different LUTs, then $size[t]$ does not depend on $size[s]$. Hence $reconvergence[s,t]$ should be set to 0. If any node i in a path from s to t is external, then s and t are in different LUTs, which gives us the following set of constraints.

$$0 \leq reconvergence[s,t] \leq m * (1 - external[i])$$

4. *Size evaluation*: If node i is external, then $size[i] = 1$ which is ensured by the following constraint.

$$1 \leq size[i] \leq m*(1-external[i]) + 1*external[i]$$

Otherwise, $size[i]$ is obtained by adding *size* variables for all fanin nodes of i, and subtracting *reconvergence* variables with i as the sink node.

$$size[i] \geq \sum_{j \in fanin[i]} size[j] - \sum_{k \in R[i]} reconvergence[k,i] - m * external[i];$$

Here $R[i]$ is the set of all source nodes that reconverge at i. In general, if there are n reconvergent paths from node k to node i, then $reconvergence[k,i]$ has to be subtracted $n-1$ times. As mentioned earlier, we have decomposed the gates such that each gate has a fanin of 2, *i.e.* $n = 2$. The *size* variable for node p in Fig. 2 is calculated below.

$$\begin{aligned} size[p] &= size[g] + size[k] - reconvergence[f,p] \\ &= 3 + 4 - 2 = 5 \end{aligned}$$

The MILP formulation for an area-minimization mapping of the graph in Fig. 2(a) is given in Fig. 4.

A key feature of our formulation is that the number of integer variables is V; these are the *external* variables. Clearly, V is the minimum number of integer variables in any formulation, since we need to assign at least one decision variable for every node. The *size* and *reconvergence* variables depend on *external* variables, and can be defined as real since they are forced to be integers in the optimal solution. There are V *size* variables, and at most E *reconvergence* variables, since a node with fanin f can be a source node in at most $f-1$ reconvergent node-pairs. Thus the total number of variables is less than $2V+E$, which is quite small.

Minimize $external[f] + external[g] + external[h] + external[k]$ **subject to :**

Boundary conditions:
$external[i] = 1, \forall\ i \in \{a,b,c,d,e,o,p\};$
$size[i] = 1, \forall\ i \in \{a,b,c,d,e\};$

Size constraints:
$1 \leq size[i] \leq 5, \forall\ i \in \{f,g,h,o,k,p\};$

Size evaluation:
$size[i] \leq 5(1-external[i]) + external[i], \forall\ i \in \{f,g,h,k,o,p\};$
$size[f] \geq size[a] + size[b] - 5\ external[f];$
$size[g] \geq size[f] + size[c] - 5\ external[g];$
$size[h] \geq size[f] + size[d] - 5\ external[h];$
$size[k] \geq size[h] + size[e] - 5\ external[k];$
$size[o] \geq size[g] + size[h] - reconvergence[f,o] - 5\ external[o];$
$size[p] \geq size[g] + size[k] - reconvergence[f,p] - 5\ external[p];$

Reconvergence evaluation:
$reconvergence[f,o] \leq 5(1-external[i]), \forall i \in \{g,h\};$
$reconvergence[f,p] \leq 5(1-external[i]), \forall i \in \{g,h,k\};$
$0 \leq reconvergence[f,o], reconvergence[f,p] \leq size[f];$

Integrality constraints:
$external[i] \in \{0,1\}, \forall\ i \in \{f,g,h,k\};$

Figure 4: MILP formulation of LUT-based FPGA mapping for Fig. 2 with $m = 5$.

Delay minimization: We define a *level* variable for every node, which is the level of the node in the LUT cover of G. The level of node i is the maximum level among all its input nodes if i is an internal node, and one more than the maximum level if it is external. If i has j and k as fanin nodes, then we use the following constraint.

$$level[i] = external[i] + max\{level[j], level[k]\}$$

We can linearize this constraint using the two inequalities given below. The optimal solution will ensure that the above constraint is satisfied.

$$level[i] \geq external[i] + level[j]$$
$$level[i] \geq external[i] + level[k]$$

The *level* variables are defined as real, since they depend on *external* variables, and thus are implicitly forced to be integer in the optimal solution.

Delay minimization with fixed area: We assume the FPGA to be an array of $N \times N$ LUTs. Area can be fixed, while minimizing delay, by adding just one constraint, *i.e.*

$$\sum_{i=1}^{V} external[i] \leq N^2$$

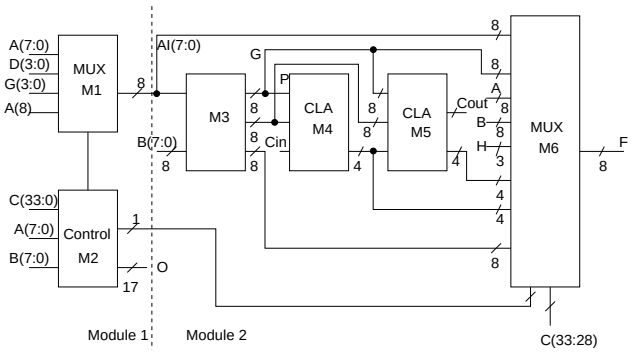

Figure 5: High-level model [10] of the c880 ISCAS85 benchmark circuit and its partition.

Circuit	No. of LUTs (execution time) using the MILP formulation		Best known no. of LUTs using prior techniques
	With partitioning	Without partitioning	
c432	$39 + 23$ $= 62$ (3 min.)	62 (30 min.)	70†
c499	$24 + 40$ $= 64$ (1.5 min.)	64 (1 hr.)	74‡
c880	$44 + 43$ $= 87$ (4 min.)	86 (2 hr.)	109‡
c6288	$31 \cdot 2 + 32 \cdot 13$ $= 478$ (2 min.)	*	478†

*:Unable to complete due to enormous circuit size
†:Obtained by running *mis-pga* ‡:Quoted from [7]

Table 1: LUT-based mapping of ISCAS-85 circuits with and without partitioning.

5 Circuit Partitioning

The execution time for our MILP approach on an IBM $RS6000$ workstation varies from a few CPU seconds for small circuits with fewer than 50 gates to about an hour for large circuits with over 500 gates. We now present a technique to reduce the computation time for very large circuits by partitioning the circuits, and mapping each subcircuit independently. The required number of LUTs remains the same or increases only marginally as a result of partitioning. Consider a partition of graph G which has a cutsize of K. The cutsize of a partition is defined as the number of nodes whose fanout edge(s) are cut by the partition. Let N_0 be the number of LUTs in the optimal cover of G, and N be the number of LUTs in the cover of G obtained after partitioning G. We can easily show that $N \leq N_0 + K$. Thus the main criterion for an efficient partition is a small cutsize.

The high-level specification of a circuit provides a natural and efficient partition for mapping purposes. We have studied the effect of partitioning on the optimality and execution time for some ISCAS-85 benchmark circuits [2], as shown in Table 1. The circuits

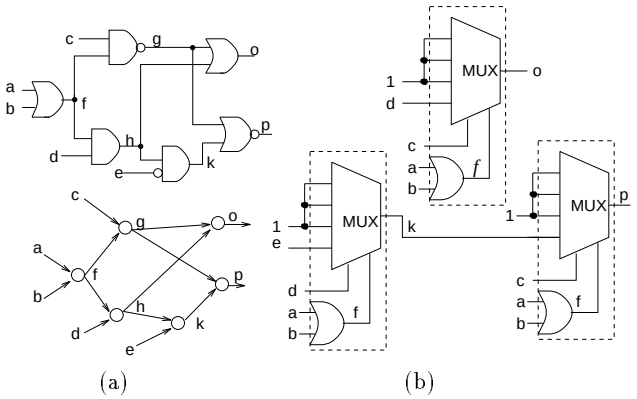

Figure 6: Mapping using $ACT1$ logic blocks: (a) circuit C and its graph G, and (c) optimal mapping using 3 logic blocks.

are partitioned according to the high-level models described in [10]. Figure 5 shows the high-level model of c880 circuit, which is an 8-bit ALU. We obtained an efficient partition of cutsize 9 for c880.

Our results show that partitioning drastically reduces the execution time while resulting in almost the same number of LUTs as the optimal cover of the unpartitioned circuit. We found that *mis-pga* [12] generates covers for c6288 of 478 and 898 LUTs with and without partitioning, respectively. This suggests that the existing heuristic techniques may not work well for very large circuits, and partitioning might significantly improve the quality of solution for these techniques.

6 Multiplexer-based FPGAs

We now describe an MILP formulation of the FPGA mapping problem for mux-based logic blocks. Since FPGAs like the Actel $ACT1$ and $ACT2$ series attach auxiliary functions (AND or OR gates) to the multiplexers, we need a very general MILP formulation that can accommodate functional information such as the types of gates used in the logic blocks. The MILP approach has the ability to handle such functional constraints.

We model a 2^n-to-1 multiplexer as a LUT of size $n+1$, where one input is restricted to be in either true or complemented form, but not both, in the function realized by the LUT. This restricted input acts as the data input of the multiplexer, while the remaining inputs are the select inputs. Consider the MILP formulation for the $ACT1$ logic block of Fig. 1. We treat the logic block as a 3-input LUT with one data input, and an optional OR gate feeding one of the other two inputs. (The OR gate was apparently included in the $ACT1$ logic block to enable the realization of a 4-input OR gate.) Figure 6 shows an optimal cover for the circuit of Fig. 2 obtained via our MILP method.

To map a circuit using $ACT1$ logic blocks, we add the following constraints and variables to the basic

MILP formulation developed in Sec. 4 (Fig. 4).

Gate-inversion calculation: Unlike the LUT-based logic blocks, we have to ensure that the inversion parity of the paths from the primary inputs to the primary outputs in the cover of G exactly match those in the input circuit C. The inversion parity of a path is 0 (1) if the path contains an even (odd) number of inversions. We assign a binary variable *inversion* to every node of G: *inversion*[i] is 0 (1), if the inversion parity of the path from the primary inputs to i in C and its cover are the same (opposite). In Fig. 6, *inversion*[k] = 1 since C and its cover have opposite inversion parity from e to k.

Data input constraint: We have to ensure that every multiplexer has at least one restricted input which can act as the data input. We introduce a variable *data-size* for every node: *data-size*[i] counts the number of restricted inputs feeding node i. The *data-size* variable is similar to the *size* variable in the LUT case, which counts the number of inputs feeding a node. If the reconvergent paths from a source node to a sink node have opposite inversion parities, then the *data-size* variable for the source node is set to 0. In Fig. 6, the paths from f to o have opposite parities, thus *data-size*[f] = 0.

OR gate mapping: We define a binary variable *or-gate* for every node, where *or-gate*[i] = 1, if i is mapped to the OR gate in an $ACT1$ logic block, and 0 otherwise. In the circuit of Fig. 6, only node f has *or-gate* = 1.

The area objective is defined as the sum of the *external* and *inversion* variables for the primary outputs, because an inverted primary output in the cover of C requires a logic block to correct its inversion parity.

7 Experimental Results

We have solved the foregoing MILP problems using the OSL package developed by IBM Corp. [11]. OSL is an optimization tool which can solve LP problems with millions of variables and constraints. We define our MILP problems in a high level modeling language, called AMPL [8]. We have written a program for generating the MILP formulation from the circuit netlist, whose time complexity is $O(V^2)$.

We have mapped various MCNC benchmark circuits on both types of FPGAs. We first perform technology-independent logic minimization using the *mis* logic synthesis package [6]. We use the *mis* script provided by Francis *et al.* [9]. We then decompose the circuit into an equivalent circuit of 2-input gates using the *tech-decomp -a2 -o2* function from the *mis* package. These preprocessing steps are performed to compare our results with those from the LevelMap algorithm [7], which uses the same steps.

LUT-based logic blocks: We assume the size of LUT to be 5, as in the Xilinx X3000 FPGAs.

Circuit	No. of 2-input gates	No. of LUTs using the MILP approach	Best known no. of LUTs [7]
apex7	201	60	76
count	112	31	31
duke2	325	128	150
misex1	49	12	15
rd84	153	11	24
vg2	72	20	24
z4ml	27	5	6
5xp1	88	24	24
9sym	201	60	61
c432	196	62	70†
c499	392	64	77
c880	347	86	109
c6288	2406	478	478†
Total		1011	1145

†: Obtained by partitioning, and applying *mis-pga*

Table 2: Number of 5-input LUTs required by various benchmark circuits.

Circuit	Area minimization		Delay minimization	
	No. of LUTs	No. of levels	No. of LUTs	No. of levels
apex7	60	9	64	5
count	31	16	58	6
vg2	20	8	38	6
5xp1	24	5	34	3

Table 3: Number of levels of 5-input LUTs required by MCNC benchmark circuits.

Area minimization: The mapping results for various MCNC benchmark circuits from our MILP approach are compared in Table 2 with the best results from prior techniques [7]. Our results are significantly better due to the inherent optimality of the MILP approach. We have also observed that the optimal solution is usually obtained in a few minutes for very large circuits, but the MILP solver takes a much longer time to verify its optimality. Therefore very good solutions can be obtained quickly without completely solving the MILP problem.

Our algorithm decomposes the nodes of G into nodes with fanin less than the LUT size m, before the technology mapping process. Therefore, we do not compare our technique with the techniques such as *mis-pga* [12], which use functional information to perform node decomposition during the technology-mapping step.

Delay minimization with fixed area: We choose a few MCNC benchmark circuits which can be mapped on an FPGA of size 8×8. We then minimize the number of levels of LUTs while mapping the circuits on the chosen FPGA. Table 3 presents some results of

Circuit	No. of 2-input gates	No. of logic blocks using the MILP approach	No. of logic blocks using *mis-pga(new)* [13]
$vg2$	72	33	30
$z4ml$	27	9	14
$5xp1$	88	37	35
$c499$	392	161	166
$c880$	347	168	159

Table 4: Number of $ACT1$ logic blocks required by various benchmark circuits.

minimizing delay with the fixed number of LUTs.

Multiplexer-based logic blocks: We map a few MCNC benchmark circuits using the $ACT1$ logic blocks, and compare our algorithm to *mis-pga* [13] in Table 4. *mis-pga* constructs a binary decision diagram (BDD) or an if-then-else directed acyclic graph for every node in the circuit graph. It then maps every BDD using $ACT1$ logic blocks, since a BDD node corresponds to a 2-to-1 multiplexer. Thus *mis-pga* performs logic synthesis and technology mapping in conjunction. On the other hand, we assume that logic synthesis, *i.e.* technology-independent minimization and node decomposition, is performed prior to our technology mapping algorithm. Nevertheless, the results produced by the MILP and *mis-pga(new)* algorithms are comparable, as shown in Table 4.

8 Conclusions and Future Directions

We have designed an efficient MILP formulation for the FPGA technology mapping problem which can generate truly optimal results. We have demonstrated that these results are considerably better than those obtained from prior heuristic techniques for various MCNC benchmark circuits. The MILP approach is highly flexible, since it can be applied to a wide range of logic block architectures. Another advantage of our approach is that the objective function can be easily modified to optimize area, delay, or a combination of both. Prior techniques, on the other hand, were aimed at a specific logic block architecture and a specific objective function. We have also shown that the execution time for very large circuits can be reduced drastically with little or no loss in optimality by partitioning the circuit, and mapping each subcircuit individually.

Our MILP formulation currently handles logic blocks that consist of a single LUT or a single multiplexer with an OR gate. We are extending it to handle logic blocks with any multi-output and multi-level arrangement of LUTs, multiplexers, AND and OR gates, *e.g.* the Xilinx $X4000$ logic block which has two levels of LUTs with two outputs. The general MILP method can be used for comparative studies of various types of logic blocks, leading to the design of efficient logic block architectures. The MILP method can also be used to select the most appropriate FPGA family for a particular circuit design.

9 Acknowledgments

We thank Mark C. Hansen for providing the netlists for various modules in the high-level models of the ISCAS benchmark circuits.

References

[1] Actel Inc. *The Actel FPGA Data Book*, Sunnyvale, Calif., 1993.

[2] F. Brglez and H. Fujiwara. A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in FORTRAN. *Proc. IEEE Int'l Symp. on Circuits and Systems*, 1985, pp. 705–712.

[3] J. Cong and Y. Ding. On Area/Depth Trade-off in LUT-based FPGA Technology Mapping. *Proc. 30th Design Automation Conf.*, 1993, pp. 213–218.

[4] J. Cong and Y. Ding. FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in LUT Based FPGA Designs. *IEEE Trans. on CAD*, vol. 13, pp. 1–11, Jan. 1994.

[5] G. B. Dantzig. *Linear Programming and Extensions.* Princeton Univ. Press, 1963.

[6] E. Detjens *et al.* Technology Mapping in MIS. *Proc. Int'l Conf. on CAD*, 1987, pp. 116–119.

[7] A. H. Farrahi and M. Sarrafzadeh. Complexity of the LUT Minimization Problem for FPGA Technology Mapping. *IEEE Trans. on CAD*, vol. 13, pp. 1319–1332, Nov. 1994.

[8] R. Fourer, D. Gay, and B. Kernighan. *AMPL Reference Manual*, 1992.

[9] R. J. Francis, J. Rose, and Z. Vranesic. Chortle-crf: Fast Technology Mapping for LUT Based FPGAs. *Proc. 28th Design Automation Conf.*, 1991, pp. 613–619.

[10] M. C. Hansen and J. P. Hayes. High-Level Test Generation using Physically-Induced Faults. *Proc. of VLSI Test Symposium*, 1995, pp. 20–28.

[11] IBM Corp. *Optimization Subroutine Library*, 1990.

[12] R. Murgai *et al.* Logic Synthesis for Programmable Gate Arrays. *Proc. 27th Design Automation Conf.*, 1990, pp. 620–625.

[13] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli. An Improved Synthesis Algorithm for Multiplexer-based PGAs. *Proc. 29th Design Automation Conf.*, 1992, pp. 380–386.

[14] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley, New York, 1988.

[15] A. Sangiovanni-Vincentelli, A. El Gamal, and J. Rose. Synthesis Methods for FPGAs. *Proc. of IEEE*, July 1993, pp. 1057–1083.

[16] Xilinx Inc. *The Xilinx FPGA Data Book*, Santa Clara, Calif., 1994.