Inverter Minimization in Multi-Level Logic Networks

Alok Jain*
Department of ECE
Carnegie Mellon University
Pittsburgh, PA 15213

Randal E. Bryant*
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

In this paper, we look at the problem of inverter minimization in multi-level logic networks. The network is specified in terms of a set of base functions and the inversion operation. The library is specified as a set of allowed permutations of phase assignments on each base function. Traditional approaches to this problem have been limited to greedy heuristics based on local information. Our approach takes a more global view and maps the problem of inverter minimization into a problem of removing a minimum of vertices from a graph, so as to make the remaining graph 2-colorable. This approach has the flexibility of capturing a variety of design-specific features that are relevant to the problem of inverter minimization. Although, in general the problem is NP-complete, we have developed several good heuristic and branch and bound search techniques.

1 Introduction

The *Technology Binding* problem is to cover an arbitrary logic network with a library of primitive cells with the intent of minimizing the cost function associated with the gates in the final representation. The binding process on DAGs has been proven to be NP-complete[1]. A restricted version of technology binding is the *inverter minimization* problem. The logic network is described in terms of a set of *base functions* and the INVERT operation. The library of primitive cells are all the base functions in the logic network along with a selected permutation of phase assignments to the input and output pins of these cells. The objective is to find a phase assignment for the logic network that minimizes the number of inverters.

Previous work done in this area includes the Dagon[1], MIS[2] and Ceres[3] systems. All these systems use partitioning algorithms to decompose the DAG, representing the logic network, into forests of trees and then perform inverter minimization at the tree-level. There have been several attempts to overcome this limitation. In the past, inverter minimization on DAG networks has been referred to as the *generalized phase assignment* problem[4]. A restricted form of generalized phase assignment is the *global phase assignment* problem, where each gate in the network can only be replaced by its dual. The global phase assignment problem was shown to be NP-complete by a transformation from MAX-CUT[4]. The MIS system incorporates two heuristics for global phase assignment on DAG networks[5]. *QuickPhase* is based on computing an *inverter savings* for each gate and selecting the gate with the maximum inverter savings for replacement by its dual. QuickPhase has a limitation in that it terminates after finding a local minimum. A partial hill climbing scheme is used in *GoodPhase* to obtain better results at the expense of CPU time.

In our approach, the library is mapped into a set of *phase-constraint graphs*. The logic network is processed to derive a *polarity graph* which states how the network differs from the phase-constraint graphs. The inverter minimization problem is mapped into a problem of eliminating a minimum number of vertices from the polarity graph, so as to make the remaining graph 2-colorable. Section 2 is divided into various subsections each of which discusses the steps in the formulation of the problem. Section 3 shows the flexibility of our approach in capturing various design features that are relevant to the task of inverter minimization. Some experimental results and comparisons with phase assignment algorithms in MIS are given in section 4.

2 Problem Formulation

2.1 The Library

The library is a set of allowed permutation of phase assignments to the input and output pins of each of the base functions. Phase assignments corresponding to a single base function are referred to as a *family of patterns*. Pins are assumed to be ordered. For n distinct base functions in the logic network, there should be n families in the library. For a family described over a base function with m pins, there can be a maximum of 2^m patterns in the family. Each family has to have the base function pattern, which corresponds to the pattern with a positive phase assigned to all the input and output pins.

A phase-constraint graph is derived for each family of patterns in the library. A vertex in the phase-constraint graph corresponds to a pin in the base function. An edge between two vertices specifies that the corresponding pins have the same phase assignment in all patterns in the family. As an example consider the AND/OR primitive cells shown in figure 1. The primitive cells correspond to a family of two patterns with the AND (OR) cell as the pattern with a positive (negative) phase assignment on all pins. The phase-constraint graph for this family has 3 vertices corresponding to the pins in the base function. The edges specify that all 3 pins should have the same phase assignment (either all positive or all negative). Note the fact that the phase-constraint graph is not unique to the set of AND/OR cells. Thus the 3-way graph also represents the constraints for the set of NAND/NOR cells when the base function is a NAND gate. Figure 2 shows the phase-constraint graph for the AND/OR/NAND/NOR prim-

*This research partially funded by Semiconductor Research Corporation contract number 92-DC-068, and by Defense Advanced Research Projects Agency, ARPA order number 4976

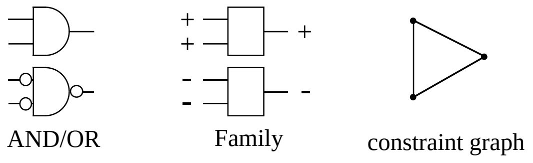


Figure 1: 3-way phase constraint

itive cells. The 2-way phase-constraint graph specifies only an input constraint. The output is unconstrained since it can have either positive or negative phase for all allowed assignments to the input pins. The constraint graphs can capture the concept of MIS global phase assignment. Constraint graph for an m-input single-output base function will be a *complete graph* with $m+1$ vertices.

2.2 The Network

The logic network is described in terms of a set of base functions and the INVERT operation. The network is mapped into a polarity graph. The polarity graph states how each gate in the logic network differs from the phase-constraint graph of the corresponding base function.

At this point, we introduce some terminology that will enable us to formulate the problem. In the logic network a net i is denoted as n_i. A gate k in the logic network is denoted as g_k. For every net n_i serving as an input to a gate g_k, let $\gamma(g_k, n_i)$ denote whether the net is fed to the gate in true ($\gamma(g_k, n_i) = +$) or complemented ($\gamma(g_k, n_i) = -$) form. For a net n_i serving as the output of gate g_k, let ($\gamma(g_k, n_i) = +$). Each net n_i in the logic network will be mapped into a vertex v_i in the polarity graph. In the polarity graph, each edge $\{v_i, v_j\}$ has a label, denoted $\lambda(\{v_i, v_j\}) \in \{+, -\}$. An operator $\bullet$ operates over the elements of the set $\{+, -\}$, such that $(+ \bullet +) = (- \bullet -) = +$ and $(+ \bullet -) = (- \bullet +) = -$.

For each net n_i in the logic network, introduce a vertex v_i in the polarity graph. For each gate g_k in the logic network, look at each edge in the phase-constraint graph of the corresponding base function. Assume that n_i and n_j are nets in the logic network corresponding to endpoint vertices in the phase-constraint graph. Introduce an edge between vertices v_i and v_j in the polarity graph with a label $\lambda(\{v_i, v_j\}) = \gamma(g_k, n_i) \bullet \gamma(g_k, n_j)$. A positive edge ($\lambda(\{v_i, v_j\}) = +$) represents the fact that the vertices v_i and v_j are required to have the same phase assignment. A negative edge ($\lambda(\{v_i, v_j\}) = -$) specifies that the vertices should have opposite phases. In general, the resultant polarity graph will be a multi-graph. Reduce the multi-graph so that two vertices have at most two edges (1 positive and 1 negative edge) between them.

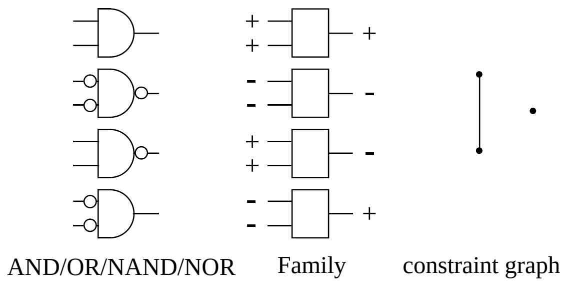


Figure 2: 2-way phase constraint

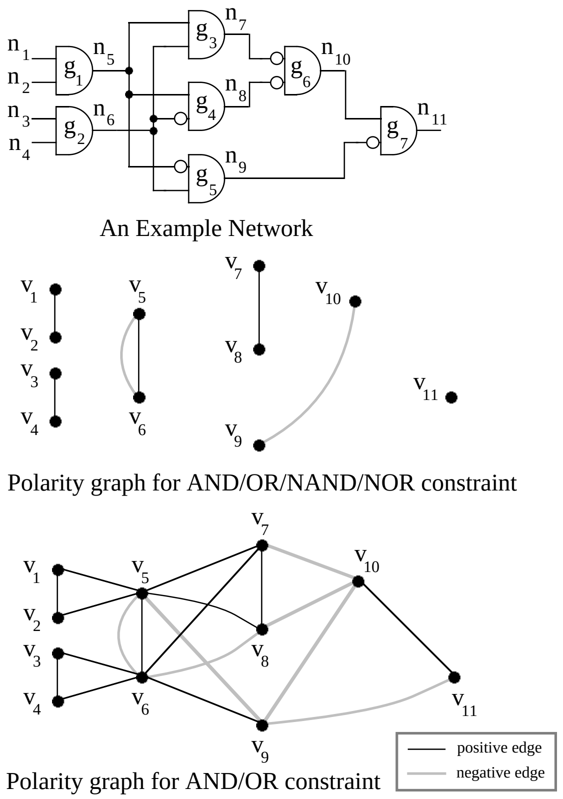
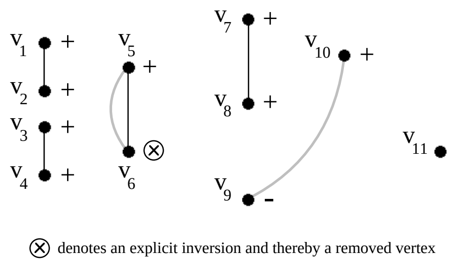


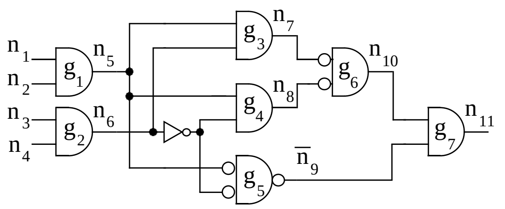
Figure 3: Example network. Bubble represents inversion.

A pair of such edges is referred to as a *double-edge*. Figure 3 shows an example network and the associated polarity graphs obtained for the AND/OR/NAND/NOR 2-way constraints and the AND/OR 3-way constraints. In the polarity graph for the AND/OR/NAND/NOR constraint, the positive edge between v_5 and v_6 introduced due to gate g_3, indicates that the vertices should be assigned the same phase in order to avoid adding any explicit inverters. The negative edge between the vertices introduced due to gates g_4 and g_5, indicates that the vertices should be assigned opposite phases in order to avoid adding any explicit inverters.

The inverter minimization problem is now reduced to a problem of assigning a unique phase to each vertex. The available phases are $\phi \in \{+, -\}$. The constraint is that the endpoints of all positive edges should be colored with the same phase and the endpoints of all negative edges should be colored with opposite phases. If the polarity graph is colorable, then the network has an inverter free representation. Note that our approach guarantees obtaining an inverter free representation if one exists for the network. In general the polarity graph may not be colorable. This implies that the logic network cannot be mapped into the library of primitive cells without the use of explicit inverters. Introducing an inverter at a net n_i implies that both phases are available for this net. The corresponding vertex v_i, along with all incident edges, can be removed from the polarity graph. Figure 4 shows one feasible coloring and the resultant gate representation for the AND/OR/NAND/NOR constraints for the example network. Note the fact that vertex v_6 has been marked for removal. The resultant gate representation has one explicit inversion. Gates g_5 and g_6 have been mapped into OR and NOR gates respectively.

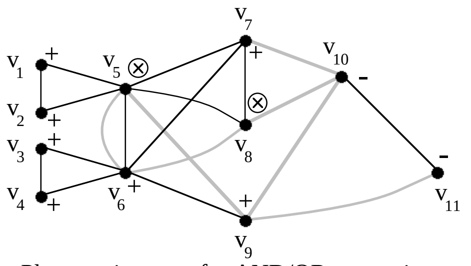

v1 +
v2 +
v3 +
v4 +
v5 +
v6 ⊗
v7 +
v8 +
v9 -
v10 +
v11
⊗ denotes an explicit inversion and thereby a removed vertex

Phase Assignment for AND/OR/NAND/NOR constraint

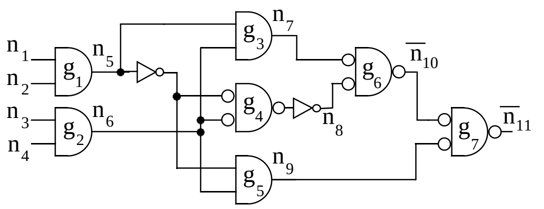


Resultant Representation for AND/OR/NAND/NOR constraint

Figure 4: 2-way constrained representation



Phase assignment for AND/OR constraint



Resultant gate Representation for AND/OR constraint

Figure 5: 3-way constrained representation

Similarly figure 5 shows a feasible coloring and the resultant gate representation for the AND/OR constraints. The gate representation has two explicit inversions. Gates g_4, g_6 and g_7 are mapped into OR gates.

2.3 Phase Assignment Algorithms

The coloring problem can be shown to be NP-complete by a transformation from the vertex cover problem[6]. This section looks at some heuristic methods and branch and bound search techniques.

Note the fact that the polarity graph is not colorable iff the graph has a cycle with an odd number of negative edges (for simplicity we will refer to these as odd cycles). The odd cycle can be broken by removing any vertex within this cycle. The *QuickColor* heuristic picks an odd cycle from the graph and then selects a vertex in the cycle with the maximum double-edge degree. If two vertices have the same double-edge degree, then select the one with the maximum edge degree. After removing the selected vertex and incident edges, attempt recoloring the remaining graph.

QuickColor picks an arbitrary odd cycle as the next candidate for vertex removal. *GoodColor* attempts to pick a "good" next odd cycle. Note that all double-edges in the polarity graph constitute an odd cycle in the graph. A prescan stage removes vertices with double-edges in the order of their double-edge degrees. For the remaining odd cycles in the graph pick the smallest cycle in the graph as the next candidate. GoodColor has an additional refinement which lets the algorithm recover from an earlier potentially "bad" choice. Each eliminated odd cycle in the graph is said to be *covered* by the vertex that was removed to break the cycle. Each vertex v_i maintains a cycle count $\aleph(v_i)$, which is the number of eliminated cycles that contain v_i. Pick the smallest cycle in the graph as the next candidate odd cycle $cycle_n$, and then select a vertex to break $cycle_n$. GoodColor uses the same edge-degree selection heuristic used in QuickColor except when $cycle_n$ shares vertices with a previously eliminated odd cycle (say $cycle_p$) which is covered by a removed vertex (say v_r) with $\aleph(v_r) = 1$. Then "undo" the removed vertex v_r and add it back to the graph. Remove one of the shared vertices which covers both cycles, $cycle_n$ and $cycle_p$. Note the fact that since $\aleph(v_r) = 1$, all previously eliminated cycles will still remain covered. This algorithm could potentially be extended to undoing vertices with greater cycle counts to resemble a hill-climbing algorithm.

For comparison purposes, a branch and bound algorithm was implemented to obtain an exact solution for the problem.

3 Other Considerations

This section looks at some enhancements and modifications to the basic formulation so as to handle various circuit details.

In certain situations system inputs or outputs might be required to be in positive or negative logic. The polarity graph can be modified to reflect this. Introduce a special vertex called the *universal-positive-phase* ($\mathcal{U}^+$). Introduce a positive (negative) edge between the $\mathcal{U}^+$ vertex and all the vertices that require positive (negative) phase. Figure 5 shows a feasible coloring and the resultant gate representation for the AND/OR constraints for the example network in figure 3, under the conditions that (i) all inputs (n_1, n_2, n_3, n_4) are specified to be in positive logic (ii) The net n_9 is an output required to be in positive logic and (iii) the output n_{11} is required to be in negative logic.

A flip-flop offers some interesting insights into the flexibility of our approach. Depending upon the design, a flip-flop can offer three distinct possibilities. (i) If the the output pin is always in positive or negative logic, then introduce the appropriate edge between the $\mathcal{U}^+$ vertex and the output vertex. (ii) If the output phase is related to the input phase, then introduce the appropriate edge between the input and output vertex. (iii) If the output is available in both true and complemented forms, then the vertex corresponding to the output terminal, can be removed from the polarity graph. Furthermore, such a flip flop does not specify any constraint on the input pin, since a negative phase assignment on the input vertex can be handled by flipping the true and complemented signals on the output.

ISCAS 85 Circuit	Two input Gates	AND map + Color heuristics / Optimal							MIS AND/OR/NAND/NOR map				
		Map	QuickColor		GoodColor		Optimal		Map	QuickPhase		GoodPhase	
		Inv	Inv	time	Inv	time	Inv	time	Inv	Inv	time	Inv	time
C1908	617	430	15	0.6	13	0.6	13	0.6	47	47	0.8	43	7.8
C2670	876	626	62	0.9	62	0.9	62	252.7	122	122	1.6	119	17.7
C3540	1265	817	20	1.2	20	1.3	20	1.3	101	100	2.0	97	32.0
C5315	2077	1379	59	2.1	58	2.2	58	452.3	233	230	4.3	197	112.3
C6288	2352	2306	0	2.7	0	2.7	0	2.8	33	32	3.4	32	102.7
C7552	2628	2038	88	2.8	88	2.8	88	3.1	365	352	8.3	331	343.9

Table 1: Experimental Results for AND/OR/NAND/NOR logic

ISCAS 85 Circuit	Two input Gates	AND map + Color heuristics / Lower Bound						MIS AND/OR map				
		Map	QuickColor		GoodColor		Optimal	Map	QuickPhase		GoodPhase	
		Inv	Inv	time	Inv	time	Inv	Inv	Inv	time	Inv	time
C1908	617	430	146	1.2	123	4.0	>106	147	145	0.9	125	19.9
C2670	876	626	180	1.7	170	6.1	>120	221	201	1.9	176	27.3
C3540	1265	817	173	2.8	169	12.0	>113	239	232	2.3	195	104.7
C5315	2077	1379	338	7.2	289	28.4	>217	370	336	6.2	298	367.8
C6288	2352	2306	503	15.4	480	15.4	>463	556	554	3.7	514	427.6
C7552	2628	2038	535	12.6	460	59.8	>345	651	555	14.7	471	1092.5

Table 2: Experimental Results for AND/OR logic

Another interesting case is a multiplexor. A 1-2 MUX specifies a 3-way constraint between the data and output pins. The MUX does not specify any constraint on the control pin. A negative phase assingment on the control pin can be handled by flipping the data terminals.

Unfortunately, there are certain families of patterns that cannot be handled by our approach. One prime example is the set of 4 patterns that constitute the XOR family.

4 Experimental Results

Tables 1 and 2 show a limited set of experimental results obtained on the ISCAS 85 benchmarks. A more comprehensive set of results can be found in [6]. All CPU times are in seconds and were obtained on a DECstation 5000. The ISCAS 85 benchmarks were first mapped into a set of two-input AND and INVERT gates by the MIS technology mapper. The Quick, Good and Optimal coloring heuristics were used for inverter minimization on this AND-logic network.

The 2-way phase-constraint graph was used to specify AND/OR/NAND/NOR cells in the library for table 1. The first and second columns is the number of two-input AND gates and inversions in the AND-logic network. The next 6 columns denote the final inverter counts and the associated CPU time for the Quick, Good and Optimal Coloring algorithms. It can be seen that the inverter count is significantly reduced. In fact for the C6288, starting from a description with 2306 inverters, we were able to find an inverter-free representation. Furthermore it can be seen that QuickColor takes almost an insignificant CPU time to give nearly optimal results. The rest of the columns in table 1 compare our coloring heuristics with phase assignment heuristics in MIS. To get the MIS numbers, the MIS technology map was used to map the ISCAS 85 benchmarks into AND/OR/NAND/NOR and INVERT cells. This step decomposed the DAG network into a forest of trees and optimized inverter counts at the tree level. QuickPhase and GoodPhase were then used to reduce the inverter count at the DAG network level. It can be seen that QuickColor beats GoodPhase by a wide margin both in terms of inverter counts and CPU times. However this is not a very fair comparison. Though the MIS technology mapper did result in a network with an optimized inverter count at the forest of trees level, QuickPhase and GoodPhase were limited to global phase assignment.

Table 2 shows experimental results obtained for the global phase assignment problem. The 3-way constraint was used to specify AND/OR cells in the library. The optimal algorithm was unable to run to completion in any reasonable amount of time on any benchmark. The reason is that 3-way constraints lead to large, dense polarity graphs. The optimal column gives a lower bound on the inverter counts. The lower bound is the number of vertex disjoint odd cycles in the graph that were found by the first iteration of the branch and bound algorithm. Comparing our results with MIS, it can be seen that QuickColor and QuickPhase give comparable results. GoodColor was able to beat or match the inverter counts in all benchmarks with a 5-20x speedup over GoodPhase.

References

[1] K. Keutzer, “DAGON: Technology Binding and Local Optimization by DAG Matching”, *24th DAC*, pp. 341-347, 1987.

[2] E. Detjens, *et al*, “Technology Mapping in MIS”, *ICCAD*, pp. 116-119, 1987.

[3] F. Mailhot, G. De Micheli, “Algorithms for Technology Mapping based on Binary Decision Diagrams and on Boolean operations”, *CSL-TR-91-486*, August, 1991.

[4] A. Wang, “Algorithms for Multilevel Logic Optimization”, *PhD Thesis, U.C. Berkeley*, April, 1989.

[5] R. K. Brayton, *et al*, “MIS: A Multiple-Level Optimization System”, *Transactions on CAD*, CAD-6(6), pp. 1062-81, November 87.

[6] A. Jain, R. E. Bryant, “Inverter Minimization in Multi-Level Logic Networks”, *CMUCAD 93-53*, August 1993.