Multi-Level Logic Optimization Using Node Complementation

Kenshu Seto, Hiroaki Yoshida, Makoto Ikeda†, Kunihiro Asada†
University of Tokyo, Department of Electrical Engineering
†VLSI Design and Education Center(VDEC)
{kenshu,hiroaki,ikeda,asada}@silicon.u-tokyo.ac.jp

1 Introduction

Multi-level logic synthesis is an important step for designing high-quality ASICs. In logic synthesis, a set of transformations is applied to an initial logic description to produce an optimized one in terms of area, delay or some other cost metric. Algebraic transformation is a fast and powerful transformation that is used in multi-level logic synthesis as a core technique. One of the other techniques used in logic synthesis is the phase assignment. Phase assignment is developed both in two-level and in multi-level logic synthesis domain. Phase assignment of two-level logic is based on the fact that for a logic function f, the complemented form $\bar{f}$ sometimes has simpler representation than the original one. On the other hand, the phase assignment technique that is well researched in multi-level logic synthesis was completely different from the one used in two-level logic synthesis. It is called as global phase assignment and it assigns a phase (original phase or complemented phase) to each node in a Boolean network to reduce the number of inverters in it. This technique is essentially different from the one used in two-level logic synthesis.

Although some observations on the application of the phase assignment technique of the two-level logic to multi-level logic were reported in the past[1], no detailed approach nor the results of it to multi-level logic optimization were published as far as authors know.

In this paper, we apply the phase assignment technique of two-level logic to multi-level logic optimization in detail. We call the application as the advanced phase assignment. An approach is developed to use the advanced phase assignment effectively. The experimental results of the new technique on area and delay optimization are reported in detail and its effectiveness is shown.

In addition, some observations are shown on the properties of the advanced phase assignment. The shortcomings of the algebraic technique is that it does not fully exploit the property of Boolean algebra. Specifically, it cannot perform Boolean decomposition which sometimes produces better logic circuits. In this paper, we show that by combining phase assignment and algebraic decomposition techniques, we sometimes obtain the Boolean decompositions of logic expressions.

2 Prerequisite

In two-level logic synthesis, a logic expression with fewer literals is chosen from the original expression and its complemented one. This operation is called phase assignment. For example, the sum-of-products (SOP) form with 11 literals

$$f = abd + abe + acd + ce$$

has a simpler expression with 8 literals when it is complemented as follows.

$$\bar{f} = \bar{a}\bar{c} + \bar{b}\bar{c} + \bar{a}\bar{e} + \bar{d}\bar{e}$$

In a Boolean network, each node has a SOP form so that the phase assignment in two-level logic can be applied to the SOP. We call this phase assignment as advanced phase assignment. For example, suppose that a node in a Boolean network has a node function as follows.

$$f = abd + abe + acd + ce$$

Then the complemented form of the function is obtained as follows as shown below.

$$\bar{f} = \bar{a}\bar{c} + \bar{b}\bar{c} + \bar{a}\bar{e} + \bar{d}\bar{e}$$

When algebraic factoring is performed to these two SOP forms, the following factored forms are obtained. A factored form of f is

$$f = a(b(d + e) + cd) + ce$$

A factored form of $\bar{f}$ is

$$\bar{f} = (\bar{a} + \bar{b})\bar{c} + (\bar{a} + \bar{d})\bar{e}$$

When De Morgan's law is applied to the factored form of $\bar{f}$, the other factored form of f is obtained as follows.

$$f = (ab + c)(ad + e)$$

The last factored form is simpler than the factored form obtained by the original, non-complemented SOP form. Besides it is one of the Boolean factored forms of the original SOP form.

There are some problems to be solved for advanced phase assignment to be effective. One is that if the SOP form of a node function is small, the advanced phase assignment does not work. For example, the complemented form of a logic function

$$f = a + b$$

is as follows.

$$\bar{f} = \bar{a}\bar{b}$$

There is no change in the number of literals between f and $\bar{f}$ in this example. Thus it is necessary to select nodes in a Boolean network and partially collapse them to make the SOP form of each node large, so that advanced phase assignment works effectively on the Boolean network. Another problem is that when a set of nodes is collapsed into one, it usually happens that the SOP form of the collapsed node becomes too large. When this problem happens, it takes a lot of time to finish advanced phase assignment. We used different strategies for partial collapsing depending on the cost metric we want to optimize while avoiding the above problems. In addition, how to select a set of nodes a priori such that collapsing the set into a node produces moderate size of SOP and complemented form of the SOP is simpler than the original SOP, is a difficult problem.

3 Properties of Factored Forms Obtained from Node Complementation and Algebraic Factoring

Node complementation followed by algebraic factoring sometimes produces a Boolean factored form of a given SOP form. In this section, some observation on the properties of Boolean factored forms obtained by the method is shown. These properties explain a class of Boolean factored forms that can be obtained by the method.

As a first property, a Boolean factored form that can be obtained by the method does not have any Boolean product in its complemented factored form. It is impossible since the method is just performing algebraic factoring of a complemented SOP and the resulting factored form cannot be a Boolean factored form. An example of a Boolean factored forms with this property is shown next. Consider the following Boolean factored form O.

$$\begin{aligned} O &= \bar{a}\bar{b}\bar{c}\bar{d}\bar{e}\bar{f} + abcdef \\ &\quad +(\bar{e}\bar{f} + \bar{c}\bar{d} + \bar{a}\bar{b})(ef + cd + ab) \end{aligned}$$

The complemented factored form of O is as follows.

$$\begin{aligned} \bar{O} &= (a + b + c + d + e + f) \\ &\quad (\bar{a} + \bar{b} + \bar{c} + \bar{d} + \bar{e} + \bar{f}) \\ &\quad ((e + f)(c + d)(a + b) \\ &\quad +(\bar{e} + \bar{f})(\bar{c} + \bar{d})(\bar{a} + \bar{b})) \end{aligned}$$

It is easy to observe that the complemented factored form $\bar{O}$ of the Boolean factored form O is also a Boolean factored form. In this case, the Boolean factored form O holds the first property and cannot be obtained by node complementation and algebraic factoring of an complemented SOP form $\bar{O}$ of O. To obtain the Boolean factored form from an SOP representation of O by the method, it is required to partition the original SOP into two disjunctive SOPs such that each SOP is having a Boolean factored form with the first property. Of course, it is not always guaranteed that we can partition a given SOP form in such a way. For the case of O, one SOP consists of cubes $\bar{a}\bar{b}\bar{c}\bar{d}\bar{e}\bar{f}$ and $abcdef$. The other SOP contains the remaining cubes of the SOP form of O. For each SOP, we expect the first property holds.

As a second property, the complemented SOP form of a given SOP form to be factored must retain a structure that is "easy to be factored". When an given SOP is complemented, the complemented SOP is minimized by two-level minimizer so that the number of cubes or literals is reduced. However an SOP form with the minimum number of cubes or literals does not always produces better factoring result. It sometimes happens that an SOP with more cubes or literals produces better factored forms when algebraic factoring is applied. It is also difficult to always guarantee the second property. We will explain second property by using an example. Let us consider following Boolean factored form P.

$$P = (\bar{e}\bar{f} + \bar{c}\bar{d} + \bar{a}\bar{b})(ef + cd + ab)$$

This Boolean factored form holds the first property, since the complemented factored form of P is written as follows and it has no Boolean product.

$$\bar{P} = (a+b)(c+d)(e+f) + (\bar{e}+\bar{f})(\bar{c}+\bar{d})(\bar{a}+\bar{b})$$

However, when the next SOP representation of P is complemented,

$$P = ab\bar{c}\bar{d} + ab\bar{e}\bar{f} + \bar{a}\bar{b}cd + \bar{a}\bar{b}ef + cd\bar{e}\bar{f} + \bar{c}\bar{d}ef$$

the following SOP form $\bar{P}_1$ is obtained.

$$\begin{aligned}\bar{P}_1 \quad = \quad & a\bar{b}c\bar{d} + ade + adf + \bar{a}b\bar{c}d + \bar{a}\bar{d}\bar{e} \\ & + \bar{a}\bar{d}\bar{f} + bce + bcf + \bar{b}\bar{c}\bar{e} + \bar{b}\bar{c}\bar{f}\end{aligned}$$

Performing algebraic factoring to the above complemented SOP $barP_1$ does not produce the desired Boolean factored form, since the complemented SOP $\bar{P}_1$ is minimized with no consideration for factoring. On the other hand, consider the following complemented SOP $\bar{P}_2$ of P.

$$\begin{aligned}\bar{P}_2 \quad = \quad & ace + acf + ade + adf + \bar{a}\bar{c}\bar{e} + \bar{a}\bar{c}\bar{f} + \bar{a}\bar{d}\bar{e} \\ & + \bar{a}\bar{d}\bar{f} + bce + bcf + bde + bdf + \bar{b}\bar{c}\bar{e} + \bar{b}\bar{c}\bar{f} \\ & + \bar{b}\bar{d}\bar{e} + \bar{b}\bar{d}\bar{f}\end{aligned}$$

Although the SOP $\bar{P}_2$ has more cubes, it is easy to be factored and it produces the desired Boolean factored form P by algebraic factoring.

4 Application of the Advanced Phase Assignment

The advanced phase assignment was implemented into the SIS environment[2]. SIS is a logic synthesis package developed at UC Berkeley. Methods to utilize advanced phase assignment for area or delay optimization are proposed in this section. Experimental results for MCNC91 benchmark circuits are shown. We performed two kinds of experiments. One is to see the factoring result obtained by the combination of node complement and algebraic factoring. The other is to see the effectiveness of advanced phase assignment on the multi-level logic optimization for area and delay.

We first utilized node complementation to improve factoring results. Specifically, we performed logic complementation and then algebraic factoring to a given SOP form to obtain its factored form. We compared the number of literals of the factored forms for each benchmark circuit with that of recent factoring algorithm[5]. The recent algorithm[5] partitions a given SOP or a product-of-sums (POS) recursively into two parts such that two parts have fewest overlaps of literals. An graph is constructed for a logic expression to examine the overlap relation of literals between each term in the expression and graph partitioning is used to find a partition of a given logic expression. In each recursive partitioning process, either an SOP or a POS of a given logic expression is chosen depending on the separability of their corresponding graphs. Each benchmark circuit is treated as a single output function by removing other outputs. Then, to obtain our factoring result for each benchmark circuit, we inverted its output and collapsed the circuit to obtain a SOP form of it. Finally, we performed quick factoring to the SOP in SIS command. The experimental result is shown in table 1. In table 1, the first column shows the names of benchmark circuits and their outputs. For example, b9_a1 represents an output function a1 of a benchmark circuit named b9. QF and GF mean quick factor and good factor of SIS command. The second and the third column represents the number of literals obtained by quick factor and good factor respectively. XF represents a factoring using graph partitioning[5] and its result is shown in the fourth column. The last column represents the number of literals obtained by our factoring algorithm using node complement and algebraic factoring. This factoring is denoted by NF. Especially, we used quick factor as an algebraic factoring. It is observed that NF gives almost the same result as XF. In some circuits, XF outperforms NF, but in some circuits NF outperforms XF. The time spent by NF is considered to be shorter than XF, since only a complementation and quick factoring is done in NF.

We then applied advanced phase assignment to area optimization. The area of a logic circuit is estimated to be the number of literals of a factored form of the logic as an approximation. As is explained in the previous section, it is necessary to collapse a set of nodes in a Boolean network to make advanced phase assignment effective but sometimes the number of literals of a collapsed node explodes. To resolve this problem, we used a SIS command `eliminate` which can perform partial collapsing of a Boolean network without increasing the number of literals. After `eliminate` is performed, we used advanced phase assignment. In SIS, there is a script called `script.rugged` which consists of strong commands to reduce the number of literals of a logic circuit. We combined advanced phase assignment with `script.rugged` to further improve the final result.

The result is shown in the table 2. The number of literals represents that of a factored form. For t481 and frg1, a large amount of reduction in literal counts is observed.

Next, we used advanced phase assignment to reduce circuit delay. Delay optimization can be done by collapsing a multi-level logic into a two-level logic. But this approach is unrealistic for a large circuit since the collapsing the circuit results in the explosion of the number of literals. In SIS, there is a script called `script.delay` for delay optimization[3]. In `script.delay`, a Boolean network is clustered into a set of nodes and each set of nodes is collapsed into two-level logic to reduce the level of the circuit with moderate increase in the number of literals. We applied advanced phase assignment to each of the partially collapsed nodes to reduce delay further. We used `lib2.genlib` as a technology library for mapping. The result is shown in the table 3. It is interesting to observe that in many circuits both delay and area are reduced by the proposed method. Especially, the optimization result for 9symml is dramatic.

In the advanced phase assignment, we choose the node function with fewer literals out of original function and its complemented one. But the circuit with fewer literals is not necessarily faster. Thus it is not guaranteed that the circuit delay is faster after the application of advanced phase assignment. To resolve this, we used the state-of-art technology mapper which is based on the mapping graph[4]. Any sets of subject graphs can be encoded into a single mapping graph and graph match process can be done on it. Besides, mapping graph can encode all the algebraic decomposition by applying specific transformations on it. This is called Δ-mapping. It is possible to encode all the algebraic decompositions of both the original SOP and its complemented SOP of each node in a Boolean network into a mapping graph. Specifically, we encoded two Boolean networks into a mapping graph for delay optimization. One of them was a Boolean network optimized only by `script.delay` and the other was optimized by the combination of `script.delay` and advanced phase assignment. The result is shown in table 4. Note that we used load-independent delay model for this last experiment, so the delay values are largely different from the previous experiment. Encouraging results are obtained with around 100

5 Conclusion

In this research, a method to optimize multi-level logic using advanced phase assignment was proposed. It was also shown that the combination of advanced phase assignment and algebraic factoring produces Boolean decompositions of logic expressions in some cases. The advanced phase assignment was experimented for area or delay optimization. It was also experimented with the state-of-art mapper using mapping graph. These experimental results were shown in detail. Although the proposed method is very simple and easy to use, its effectiveness was shown both on area and delay optimization with moderate CPU time increase.

References

[1] A. R. Wang, "Algorithms for Multi-Level Logic Optimization," Ph.D dissertation, University of California at Berkeley, May, 1989.

[2] R. K. Brayton, et al., "MIS: A multiple-level logic optimization system," IEEE Trans. CAD, CAD-6(6), pp.1062-1081, Nov. 1987.

[3] H. J. Touati, et al., "Delay Optimization of Combinational Logic Circuits by Clustering and Partial Collapsing," Proc. ICCAD, pp.188-191, Nov., 1991.

[4] E. Lehman, et al., "Logic Decomposition During Technology Mapping," IEEE Trans. CAD, vol. 16, no. 8, Aug. 1997.

[5] M. C. Golumbic, et al., "Factoring Logic Functions Using Graph Partitioning," Proc. ICCAD, pp.195-199, Nov., 1999.

circuit	number of literals			
	QF	GF	XF	NF
i2	228	228	209	209
mux	79	79	47	47
b9_a1	12	12	13	12
b9_d1	33	22	24	17
b9_i1	14	14	12	12
b9_z0	19	19	20	22
c8_r0	24	24	20	20
c8_s0	26	26	22	22
c8_t0	28	28	24	24
cmb_r	37	37	12	12
alu2_k	143	140	97	106
alu2_l	345	317	263	280
alu2_o	181	168	69	110
alu4_o	148	147	97	112
alu4_p	360	348	263	246
cm85a_l	26	26	16	19
cm85a_m	17	17	17	17
f51m_45	42	42	42	40
f51m_46	34	34	31	31
f51m_47	24	22	23	22
frg1_d0	119	111	43	42
cm162a_o	16	16	12	12
cm162a_p	36	18	15	15

Table 1: Experimental result (*factoring*)

circuit	number of literals			CPU time	
	rugged	*adv*	%	*rugged*	*adv*
9symml	186	170	8.6	5.2	4.0
C880	415	407	1.9	1.7	1.5
C1908	540	526	2.6	6.6	5.7
C2670	737	735	0.3	5.0	15.6
C3540	1296	1238	4.5	13.2	12.6
C7552	2315	2287	1.2	46.5	73.2
dalu	979	955	2.5	58.8	23.1
frg1	136	58	57.4	1.9	0.1
frg2	782	741	5.2	11.1	8.1
i3	188	136	27.7	0.4	14.6
lal	104	102	1.9	0.4	0.7
rot	671	655	2.4	3.6	3.2
t481	881	50	94.3	47.7	19.4
term1	170	151	11.2	2.8	2.1
ttt2	219	195	11.0	0.8	0.7
x4	391	388	0.7	1.2	1.3
average			14.6		

Table 2: Experimental result (*script.rugged*)

circuit	area			delay			CPU time	
	delay	*adv*	%	*delay*	*adv*	%	*delay*	*adv*
9symml	675	178	73.6	9.56	8.59	10.1	3.6	2.3
C1355	1602	1574	1.7	18.97	18.35	3.3	14.8	14.9
C2670	2309	2245	2.8	22.89	22.35	2.4	52.3	58.6
C3540	3490	3569	-2.3	30.58	29.92	2.2	94.6	97.4
c8	360	393	-9.1	7.03	6.51	7.4	1.8	1.9
cht	548	424	22.6	5.17	5.10	1.4	1.9	2.0
comp	380	340	10.5	11.72	8.81	24.8	8.8	8.6
f51m	379	344	9.2	8.46	8.37	1.1	1.6	1.5
frg1	321	115	64.2	9.98	7.76	22.2	1.6	1.5
frg2	2544	2367	7.0	11.90	10.90	8.4	27.1	26.8
rot	1895	1812	4.4	13.74	13.62	0.9	14.4	14.1
t481	2564	2496	2.7	15.41	15.29	0.8	205.1	212.3
term1	541	578	6.8	10.23	9.22	9.9	6.7	7.1
ttt2	509	909	-78.6	8.91	7.76	12.9	2.3	4.2
unreg	325	335	-3.1	5.11	4.99	2.3	1.2	1.1
x4	1154	2312	-100.3	9.22	8.00	13.2	6.6	11.7
z4ml	162	328	-102.4	7.02	6.82	2.8	0.7	1.4
average			-5.6			7.9		

Table 3: Experimental result (*script.delay*)

circuit	area			delay			CPU time	
	delta	*adv*	%	*delta*	*adv*	%	*delta*	*adv*
9symml	557	198	64.5	5.14	4.09	20.4	2.3	4.4
apex7	814	772	5.2	5.23	4.50	14.0	1.6	4.0
c8	409	334	18.3	3.37	3.09	8.3	1.1	2.4
cht	398	396	0.5	2.34	1.97	15.8	1.1	2.6
cm150	175	123	29.7	3.73	2.99	19.8	0.6	1.0
cm151	94	120	-27.7	3.35	2.72	18.8	0.4	0.6
mux_c1	62	62	0.0	2.67	2.50	6.4	0.3	0.6
cm162	153	150	2.0	3.59	3.28	8.6	0.5	0.8
cm163	130	117	10.0	3.15	2.76	12.4	0.4	0.7
comp	497	624	25.6	5.49	4.81	12.4	8.1	16.7
frg1	347	138	60.2	6.24	4.53	27.4	1.2	2.4
i7	1885	1558	17.3	2.20	1.92	12.7	11.4	30.5
mux	191	127	33.5	3.95	3.49	11.6	0.6	1.1
my_adder	739	752	-1.8	9.37	8.45	9.8	1.4	3.4
pm1	143	144	-0.7	2.39	1.98	17.1	0.5	0.8
term1	590	592	-0.3	5.44	4.17	23.3	5.6	12.8
average			14.8			14.9		

Table 4: Experimental result (Δ-mapping)