Efficient output phase assignment algorithm for PLAs

W.-J. Hsu
W.-Z. Shen

Indexing terms: Logic optimisation, Logic arrays, Phase assignment

Abstract: To implement a multiple output function, one has the option to realise each output with either true logic or complementary logic following with an inverter. In this paper, we propose an efficient algorithm to solve this output phase assignment problem for PLA implementation. Instead of using the double-phase cover minimisation approach, we use a property-checking procedure to estimate the cost of assignments. With the estimated costs, an assignment with minimum cost is chosen. The experimental results show that the proposed algorithm can obtain excellent assignment compared with other approaches.

1 Introduction

Owing to the regularity of structure and flexibility of programming, the PLA has become one of the most popular structures for the implementation of logic functions. However, direct implementation of logic functions with a PLA is sometimes wasteful and inefficient owing to a large number of product terms. To optimise the area and performance of a PLA many strategies have been developed [1], such as logic optimisation, partition, folding, etc. Among them, logic optimisation for PLA design has been investigated for many years and most of researches focus on the minimisation of logic functions. Significant works are MINI[2], Espresso_II[3], Espresso_MV[4], etc.

In addition to logic minimisation, another optimisation strategy named output phase assignment was proposed to further improve the performance of PLA. Given a multiple output function, one has the option to realise either true logic or complementary logic following with an inverter. With proper selection of output phase, a significant reduction of hardware cost can be achieved. A PLA with and without phase assignment is shown in Fig. 1. One can see that both inverting and noninverting buffers are used at the output. With output phase assignment, the number of product terms is reduced by one. Up to now, there are two algorithms that have been reported to solve the output phase assignment problem [5, 6]. The results of these algorithms have demonstrated the significant improvement to be had by introducing output phase assignment into PLA design. However, in these algorithms, a procedure called double-phase cover minimisation is used as a major operation. For an n-input m-output function, the double phase cover is an n-input $2m$-output function which is generated by adding the complement of an output as a new output. Experiments have shown that the execution time required for double-phase cover minimisation is much longer than the time for original cover minimisation. For some PLAs, double-phase cover minimisation may take more than four times the time required for onset cover minimisation. To avoid this time-consuming procedure without degrading the assignment results, we propose a new algorithm which

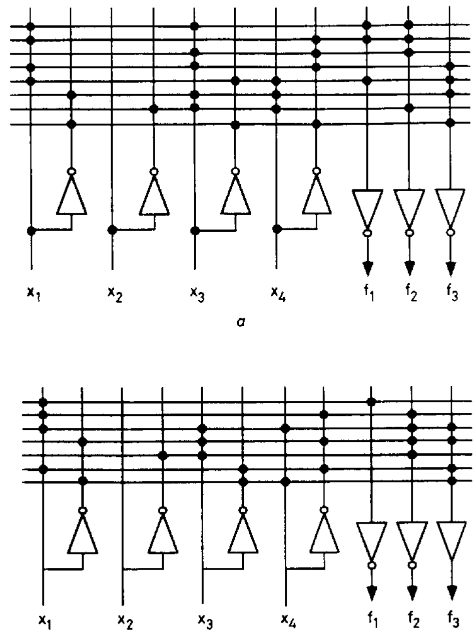

Fig. 1 *Structure of PLA*
a without output phase assignment
b with output phase assignment

This work was supported by the National Science Council, Republic of China, under grant NSC79-0404-E009-24

Paper 9570G (E10), first received 13th October 1992 and in revised form 21st April 1993

W.-J. Hsu is with the Institute of Electronics, National Chiao Tung University, Hsin-Chu 30050, Taiwan, Republic of China

Prof. W.-Z. Shen is with the Department of Electronics Engineering & Institute of Electronics, National Chiao Tung University, Hsin-Chu 30050, Taiwan, Republic of China

contains a cube-examining procedure. The proposed algorithm has been implemented on Sun workstation in C language to demonstrate its performance. Among the benchmarks described in Reference 3, many examples which Sasao's approach [5] fails to perform phase assignment within 50000 seconds are optimised with our algorithm within 3000 seconds. In addition to speed improvement, the total number of product terms after output phase optimisation is also the least in comparison with other algorithms.

2 Fundamental concepts

Given a n-input m-output logic function one can represent each output function in sum-of-product form. For simplicity, each product term in the sum-of-product form is represented with cube notation. A **cube** is a $(n+m)$-tuples vector. The first n-tuples denote the conjunction of input variables and is written as a bit vector with each bit position representing a distinct variable. The value taken by each bit can be 1, 0 or 2 (don't care), signifying the true form, negated form and nonexistence of the variable corresponding to that position, respectively. The last m-tuples denote the outputs where this product term appears, each tuple takes the value 1 or 0, signifying the product term appears or does not appear in the corresponding output. A **minterm** is a cube with only 0 or 1 entries in the first n-tuples. Given a cube c the input parts of cube is denoted as $I(c)$, and the output parts is denoted as $O(c)$. Cube c can be denoted as $I(c) \bullet O(c)$, where symbol $\bullet$ denotes concatenation operation of two vectors.

The Boolean operations of two input vectors are defined as

AND

$\wedge$	0	1	2
0	0	$\varnothing$	0
1	$\varnothing$	1	1
2	0	1	2

OR

$\vee$	0	1	2
0	0	2	2
1	2	1	2
2	2	2	2

The symbol $\varnothing$ denotes null variable. The Boolean operations of two output vectors are defined as

AND

$\wedge$	0	1
0	0	0
1	0	1

OR

$\vee$	0	1
0	0	1
1	1	1

XOR

$\oplus$	0	1
0	0	1
1	1	0

The complement ($\sim$) operation of output vector is defined as: $\sim(0) = 1$, $\sim(1) = 0$. A **cover** is a set of cubes. **Onset** cover F is the set of cubes which set the output to 1. **Offset** cover R is the set of cubes which set the output to 0. **Don't care set** cover DC is the set of cubes which can be omitted. Output vector e_1 is said to be contained in output vector e_2 if for any bit $e_1 = 1$ implies the corresponding bit of e_2 is also 1, and is denoted as $e_1 < e_2$. Cube $\boldsymbol{c1}$ is said to be covered by cube $\boldsymbol{c2}$ if every minterm of $\boldsymbol{c1}$ is contained in cube $\boldsymbol{c2}$, and this is denoted as $\boldsymbol{c1} \subset \boldsymbol{c2}$. Cube $\boldsymbol{c1}$ is said to be covered by cover F if every minterm of $\boldsymbol{c1}$ is contained in cover F, and this is denoted as $\boldsymbol{c1} \subset F$.

Definition 1: Phase vector v: Let v be a m-tuple vector, and is represented as $v = (v_1, \ldots, v_m)$, where $v_i \in \{0, 1\}$, for $i = 1$ to m. v is defined as a phase vector such that $v_i = 1$ when the ith output f_i of function f is assigned to be in complementary phase, and $v_i = 0$ when f_i is in true phase.

Definition 2: Phase cost $t(v, f)$: Let $t(v, f)$ denote the minimum number of product terms required to implement logic function f with output phase vector $\boldsymbol{v}$.

With definitions 1 and 2 the problem of output phase assignment can be restated: Given an m-output logic function f, find a phase vector v for f such that the cost $t(v, f)$ is minimum.

Given a PLA with onset cover F, offset cover R and phase vector v, the onset cover $F_a(v)$ of the PLA with phase assignment is formed as follows

(a) In F cover, cubes which contribute to the ith output for some i with $v_i = 0$ are appended to $F_a(v)$. For these cubes, outputs with respect to $v_i = 1$ are set to 0.

(b) In R cover, cubes which contribute to the ith output for some i with $v_i = 1$ are appended to $F_a(v)$. For these cubes, outputs with respect to $v_i = 0$ are set to 0.

Example 1: Given minimised onset and offset covers F_m, R_m of a logic function with four-input three-output, the onset cover F_a of PLA with phase vector $v = (110)$ is

	F_m		R_m		$F_a\,(v = 110))$
c_0	1211 001	d_0	0200 001	c_0	1211 001
c_1	2100 010	d_1	0211 001	c_2	1200 001
c_2	1200 001	d_2	1020 010	c_3	0210 001
c_3	0210 101	d_3	1201 001	c_4	0201 001
c_4	0201 001	d_4	1210 001	d_2	1020 010
c_5	2201 010	d_5	2210 010	d_5	2210 010
c_6	0221 110	d_6	1212 010	d_6	1212 010
c_7	0202 110	d_7	1222 100	d_7	1222 100

The initial cost for $F_a(v = (110))$ is 8.

After logic minimisation, we know that given a phase vector v the minimum cost $t(v, f) \leqslant$ the number of cubes in $F_a(v)$. If one can find out how the cubes in $F_a(v)$ are merged, one may obtain a value close to $t(v, f)$. Taking these approximate numbers as a merit of implementation cost, one can assign the phase of outputs effectively. After investigating the relation between cubes, we summarise two conditions that two cubes can merge into one cube with output phase assigned properly.

(i) If the input parts of two cubes can be reduced to be the same, these cubes can merge.

(ii) If two cubes can merge for a subset of outputs, a phase assignment which excludes outputs outside the subset will make these cubes merge.

Example 2: In Example 1, cube (1212 010) in R_m can be reduced to be (1211 010) without affecting the functionality of R_m. This cube can merge with cube (1211 001) in F_m when phase vector is (010) or (110). Considering another condition, cube (0210 101) and cube (0202 110) in F_m can merge into (0222 100) with respect to the first output, then for phase vector (011) the two cubes will merge into (0222 100).

These two conditions can be formulated as the following properties:

Property 1: For cubes $\boldsymbol{c} \in F$, $\boldsymbol{d} \in R$, where $I(\boldsymbol{c}) \wedge I(\boldsymbol{d}) \neq \varnothing$, if $\boldsymbol{c} \subset (F - \boldsymbol{c} + (I(\boldsymbol{c}) \wedge I(\boldsymbol{d}) \bullet O(\boldsymbol{c}))$ and $\boldsymbol{d} \subset (R - \boldsymbol{d} + (I(\boldsymbol{c}) \wedge I(\boldsymbol{d})) \bullet O(\boldsymbol{d}))$ then cubes $\boldsymbol{c}$ and $\boldsymbol{d}$ can merge when phase vector v satisfies $(\sim v) \wedge O(\boldsymbol{c}) \neq 0$ and $v \wedge O(\boldsymbol{d}) \neq 0$.

Property 2: For cubes $\boldsymbol{c1}, \boldsymbol{c2} \in F$, where $O(\boldsymbol{c1}) \wedge O(\boldsymbol{c2}) \neq O$, let e be a m-tuple vector, $e < O(\boldsymbol{c1}) \wedge O(\boldsymbol{c2})$. If $(I(\boldsymbol{c1}) \vee I(\boldsymbol{c2})) \bullet e \subset F$, then cubes $\boldsymbol{c1}$, $\boldsymbol{c2}$ can merge when the assigned phase vector v satisfies $(\sim v) \wedge e \neq 0$ and $((\sim v) \wedge ((O(\boldsymbol{c1}) \vee O(\boldsymbol{c2})) \oplus e) = 0$.

Property 3: For cubes $\mathbf{c1}, \mathbf{c2} \in R$; $O(\mathbf{c1}) \wedge O(\mathbf{c2}) \neq O$, let e be a m-tuple vector, $e < O(\mathbf{c1}) \wedge O(\mathbf{c2})$. If $(I(\mathbf{c1}) \vee I(\mathbf{c2})) \bullet e \subset R$, then cube $\mathbf{c1}$, $\mathbf{c2}$ can merge when the assigned phase vector v satisfies $v \wedge e \neq 0$ and $v \wedge ((O(\mathbf{c1}) \vee O(\mathbf{c2})) \oplus e) = 0$.

Example 3: For the PLA given in Example 1, the following pairs of cubes satisfy the described properties:

c_0 and d_6 which satisfy property 1, can merge when phase vector is (010) or (110)

c_5 and d_3 which satisfy property 1, can merge when phase vector is (001) or (101)

c_6 and d_1 which satisfy property 1, can merge when phase vector is (001), (011) or (101)

c_3 and c_7 which satisfy property 2, can merge when phase vector is (011).

Based on these properties, an algorithm for output phase assignment is proposed and implemented.

3 Phase determination algorithm

The flow of the proposed assignment algorithm is shown as follows:

```
Algorithm 1: Output phase assignment algorithm
/* Input: onset cover F (+ don't care set DC) */
/* Output: onset cover F after phase assignment */
R = complement(F,DC); compute offset cover
If (DC set is not empty)
   F = complement(R,DC); recompute onset cover
F1 = minimise(F,R,DC); minimise onset cover
R1 = minimise(R,F,DC); minimise offset cover
If (no. of output < default_size) /* default_size is 12 in experiment */
   phase = find_minimum_phase(F1,R1);
else
   phase = find_near_minimum_phase(F1,R1);
(F_a,R_a) = phase_setup(phase, F,R);
F_m = minimise(F_a,R_a);
return(F_m)
```

After reading in a PLA, its complement the offset cover R is generated. If a DC-set exists, F is recomputed from R and DC to ensure the mutually disjoint property among F, R and DC covers. This restriction is required when minimising the offset cover R. After these preprocessing works, both F and R are minimised individually. With these minimised covers, the properties described in the previous Section are checked for cost estimation. Because the proposed algorith is a heuristic algorithm and intends to improve the speed performance for large PLA, it does not guarantee the best solution. To compromise between optimisation quality and execution time, the procedure for cost estimation and phase assignment is divided into two parts: for small PLA, the algorithm records all phase vectors when estimating the cost. And for large PLA, the algorithm stores the mergiability for each pairs of outputs. When the phase of outputs are determined, the algorithm invokes phase_setup subroutine to generate the onset for the selected phase vector. After minimising the phase assigned PLA, the output phase optimisation procedure is terminated.

The best way for recording the checking results is to update the estimated cost for each phase vector v. However, when the number of outputs is large, the number of phase vectors will be unreasonably large. It is impractical to record all possible phase vectors. To compromise between complexity and optimisation quality of the algorithm, the way that the checking results are recorded is different and depends on the number of outputs.

3.1 Minimum cost assignment algorithm

When the number of outputs is small, an integer array **phase_wei** is allocated to store the estimated cost for each phase vector.

Definition 3: index of a phase vector v — **index(v)**. Given a phase vector v, the index(v) is defined as the value of binary representation of v with v_l as LSB and v_m as MSB.

For example, index of phase vector $v = (110)$ is **index(v)** $= 3$. With **index(v)**, each phase vector v is mapped to one content of array **phase_wei**. The contents of array **phase_wei** are calculated with the following steps:

A1 For each cube $\mathbf{c}$ in F_m, if $(\sim v) \wedge O(c) \neq 0$

phase_wei(index(v)) = phase_wei(index(v)) $+ 1$.

A2 For each cube $\boldsymbol{d}$ in R_m, if $v \wedge O(\boldsymbol{d}) \neq 0$

phase_wei(index(v)) = phase_wei(index(v)) $+ 1$.

A3 For each pair of cubes $\boldsymbol{c}$ and $\boldsymbol{d}$ which satisfy property 1, if $(\sim v) \wedge O(\boldsymbol{c}) \neq 0$ and $v \wedge O(\boldsymbol{d}) \neq 0$

phase_wei(index(v)) = phase_wei(index(v)) $- 1$.

A4 For each pair of cubes $\mathbf{c1}$ and $\mathbf{c2}$ that satisfy property 2, if $(\sim v) \wedge e \neq 0$ and $(\sim v) \wedge ((O(\mathbf{c1}) \vee O(\mathbf{c2})) \oplus e) = 0$

phase_wei(index(v)) = phase_wei(index(v)) $- 1$.

A5 For each pair of cubes $\boldsymbol{d1}$ and $\boldsymbol{d2}$ that satisfy property 3, if $(v \wedge e) \neq 0$ and $v \wedge ((O(\boldsymbol{d1}) \vee O(\boldsymbol{d2})) \oplus e) = 0$

phase_wei(index(v)) = phase_wei(index(v)) $- 1$.

The first and second steps calculate the initial costs for all phase vectors. Step 3–5 then adjusts the costs according to the checking results. During property checking, a cube is marked if it satisfies any of the three properties. This will restrict each cube to merge with other cubes at most one time. After these five steps, contents of the array **phase_wei** store the estimated cost for all possible phase vectors. The phase of outputs are assigned according the contents of the array.

Example 4: The estimated costs for PLA given in Example 3 are listed as:

phase vector:	(000)	(100)	(010)	(110)	(001)	(101)	(011)	(111)
index of array:	0	1	2	3	4	5	6	7
after step 1, 2:	8	9	9	8	9	9	10	8
after step 3:	8	9	8	7	7	7	9	8
after step 4:	8	9	8	7	7	7	8	8

After constructing the **phase_wei** array, output phase is assigned based on the value of **phase_wei**.

Theorem 1: For any phase vector v of function f

$$t(v, f) \leqslant \textbf{phase\_wei(index}(v))$$

Proof: After logic minimisation, the minimised cover must contain no redundant cube and every cube is maximised such that no two cubes can merge. Given a PLA and phase vector v, the initial onset cover of PLA with phase assignment is $F_a(v)$, the initial value of **phase_wei**(v) equals to the number of product terms in $F_a(v)$. If two cubes c, d in $F_a(v)$ satisfy any of the properties, then c and d can be replaced with single cube. The number of product terms in $F_a(v)$ is reduced by one, and the value **phase_wei**(v) is also decreased by one. When all pairs of cubes which satisfy the proposed properties are replaced by related cubes, the number of cubes in the final cover is equal to **phase_wei**(v). However, if the number of product terms in minimised cover is larger than **phase_wei**(v), then the minimised cover can be replaced by the newly formed $F_a(v)$. After logic minimisation, the number of product terms must be less than or equal to **phase_wei**(v).

For PLAs with small number of output, the estimated costs for each phase vector are stored in the array **phase_wei**. With Theorem 1, the value in array **phase_wei** provides an upper bound for phase assignment cost. The phases vector v with minimum weight is selected as the desired phase.

Example 5: For the array **phase_wei** shown in Example 4, **phase_wei**(3) is the first one that has minimum value. Therefore, the assigned phase vector is (1, 1, 0) which means that the first and second outputs are in complemented phase, while the third output is in true phase.

3.2 Near minimum cost assignment algorithm

When the number of outputs is large it is impractical to record weights for all phase vectors simultaneously. Therefore a near optimum strategy is used. Instead of one-dimensional array for all phase vectors, the algorithm uses a $2m * 2m$ matrix for storing the mutual relation between outputs. The contents of weight matrix M are formed with the following steps

B1 Let $O(c)^i$ denote the ith bit in the output parts of cube c. For any cube c in onset cover F, if $O(c)^i = 1$ then

$$M(i, i) = M(i, i) + 1;$$

B2 For any cube d in offset cover R, if $O(d)^i = 1$, then

$$M(j, j) = M(j, j) + 1, \text{ where } j = i + m;$$

B3 Let $|O(c)|$ denote the number of '1' in the output parts of cube c. For any cube c in onset cover F that does not satisfy any of the properties, if $O(c)^i = 1$ and $O(c)^j = 1$, where $i \neq j$, then

$$M(i, j) = M(i, j) - 2/|O(c)|;$$

B4 For any cube d in offset cover R that does not satisfy any of the properties, if $O(d)^i = 1$ and $O(d)^j = 1$, where $i \neq j$, then

$$M(l, n) = M(l, n) - 2/|O(d)|, \quad \text{where } l = i + m, n = j + m;$$

B5 For any pair of cubes c in F and d in R that satisfy property 1, if $O(c)^i = 1$, $O(d)^j = 1$, then

$$M(i, l) = M(i, l) - 2/(|O(c)| + |O(d)|), \quad \text{where } l = j + m;$$

B6 For any pair of cubes c, d in F, assume that there exists a m-tuple vector e, $e < O(c) \wedge O(d)$ such that e satisfy property 2. Let g be a m-tuple vector, $g = e \oplus (O(c) \vee O(d))$. For any $e^i = 1$, $g^j = 1$,

$$M(l, n) = M(l, n) - 1/(|O(e)| * |O(g)|), \quad \text{where } l = i + m, n = j + m;$$

B7 For any pair of cubes c, d in R, assume that there exists a m-tuple vector e, $e < O(c) \wedge O(d)$ such that e satisfy property 3. Let g be a m-tuple vector, $g = e \oplus (O(c) \vee O(d))$. For any i, j, $e^i = 1$, $l = i + m$, $g^j = 1$, $n = j + m$

$$M(l, n) = M(l, n) - 1/(|O(e)| * |O(g)|);$$

B8 Because the matrix is symmetrical, the lower-left contents of matrix are filled as follows:

$$M(j, i) = M(i, j) \text{ for all } j > i, \text{ where } i, j \leqslant 2m.$$

In these steps, the subtracted value is determined in such a way that the summation of subtracted value is one when two cubes can merge for a given phase vector. After these steps are done for all cubes, the weight matrix M gives the number of product terms for each output and an approximate number of cubes which can be shared between any pair of outputs. Based on this matrix, a near optimum phase determination procedure is applied to assign the phase of outputs.

Example 6: Taking the PLA in example 1 as an example, the weight matrix M is

1	2	3	4	5	6
3.0	−1.67	−1.0	0	−0.5	−1.17
−1.67	4.0	0	0	0	−1.67
−1.0	0	4.0	0	−1.0	0
0	0	0	1.0	0	0
−0.5	0	−1.0	0	3.0	0
−1.17	−1.67	0	0	0	4.0

Definiton 4: Extended phase vector u. Given a phase vector v, the extended phase vector u is a $2m$-tuples vector and is defined as:

$$u_i = \begin{cases} \bar{v}_i & \text{for } i \leqslant m \\ v_{i-m} & \text{for } i > m \end{cases}$$

Definition 5: Phase weight $w(v)$. For a given phase vector v and weight matrix M, the phase weight of v in terms of extended phase vector u is defined as

$$w(v) = \sum_{i=1}^{2m} \sum_{j=1}^{2m} u_i * u_j * M(i, j)$$

The assignment of output phase is equivalent to find a phase vector v such that the phase weight $w(v)$ is minimum. However, when the number of outputs is large, it is impossible to calculate all phase weights. To solve the problem efficiently, a near optimum assignment algorithm shown subsequently is used which is similar to the algorithm proposed in Reference 5.

Algorithm 2: near minimum assignment algorithm
/* Input: weight matrix M */
/* Output: assigned phase vector */

(i) for ($i = 1$ to $2m$) $sm(i) = \sum_{j=1}^{2m} M(i, j)$;
(ii) $k = \text{Argmax}\,(|sm(k) - sm(k+m)|)$ /* Argument of which $|sm(k) - sm(k+m)|$ is maximum. */
(iii) If ($sm(k) > sm(k+m)$)
$v_k = 1$;
clear the (k)th column of matrix M;

Table 1: Comparisons of various output phase assignment algorithms

PLA name	cube number					
	True_PLA	Com_PLA	Playground	Sasao	Proposed	Exhaustive
add6	355	387	293	293	293	293
adr4	75	83	61	61	61	61
alu1	19	20	15	15	15	15
alu2	68	40	43	40	37	37
alu3	66	49	47	37	37	37
apla	25	26	25	22	25	21
bc0	179	216	179	185	179	*
bca	180	190	180	*	180	*
bcb	156	170	156	*	156	*
bcc	137	161	137	*	137	*
bcd	117	139	117	117	117	*
chkn	140	171	138	141	136	134
co14	14	92	14	14	14	14
cps	163	147	*	153	147	*
dc1	9	11	9	9	9	9
dc2	39	43	35	37	37	35
dist	123	122	104	109	106	103
dk17	18	16	15	18	16	14
dk27	10	10	9	9	9	9
dk48	21	14	14	19	13	13
exep	109	97	97	*	97	*
f51m	77	76	76	76	76	76
gary	107	114	107	107	107	*
in0	107	114	107	107	107	*
in1	106	144	106	106	106	*
in2	136	125	125	136	116	116
in3	74	112	74	79	74	*
in4	212	242	212	224	212	*
in5	62	150	62	62	62	*
in6	54	118	54	54	54	*
in7	54	66	43	43	33	*
jbp	122	144	116	*	122	*
misg	69	51	34	34	34	*
mish	82	76	*	*	53	*
mlp4	128	131	112	111	117	110
opa	79	76	76	79	75	75
radd	75	83	61	61	61	61
rckl	32	33	32	32	32	32
rd53	31	32	22	22	22	22
rd73	127	127	93	93	93	93
risc	29	20	20	27	20	20
root	57	59	49	49	48	48
sqn	38	33	32	32	33	32
sqr6	49	43	39	42	41	39
tial	581	393	359	359	359	*
vg2	110	174	110	110	110	110
wim	9	9	8	8	8	8
x1dn	110	159	110	110	103	103
x6dn	82	161	82	82	82	82
x9dn	120	159	107	116	104	104
z4	59	59	45	45	45	45
Z5xp1	65	63	60	64	58	58
Z9sym	86	72	72	86	72	72
Total	5152	5622	4516	4603	4460	

else

$v_k = 0$;

clear the $(k+m)$th column of matrix M;

(iv) If (any of the outputs is not assigned)
goto 1;

Example 7: Given the weight matrix M in Example 6, the procedure of phase assignment is as follows:

1-1 The summation of rows:

$sm(1)$	$sm(2)$	$sm(3)$	$sm(4)$	$sm(5)$	$sm(6)$
-1.33	0.66	2.0	1.0	1.5	1.16

1-2 Because $|sm(1) - sm(4)| = 2.33$ is maximum, select $k = 1$.

1-3 $sm(1) < sm(4)$, let $v_1 = 0$, the first output is set to be in true phase. Clear the contents in the fourth column.

2-1 Recalculate the summation of each row

$sm(1)$	$sm(2)$	$sm(3)$	$sm(4)$	$sm(5)$	$sm(6)$
*	0.66	2.0	•	1.5	1.16

Table 2: Comparisons of execution time for phase assignment algorithms

PLA name	No_ Input	No_output	Execution time Sasao	Proposed	speedup
add6	12	7	22095	12222	1.81
adr4	8	5	1045	551	1.90
alu1	12	8	137	43	3.19
alu2	10	8	1503	734	2.05
alu3	10	8	2317	881	2.63
apla	10	12	918	1197	0.77
bc0	26	11	15694	15198	1.03
bcd	26	38	352460	468405	0.75
chkn	29	7	17483	16022	1.09
co14	14	1	169	141	1.20
cps	24	109	859428	327818	2.62
dc1	4	7	43	35	1.23
dc2	8	7	383	369	1.04
dist	8	5	3042	1525	1.99
dk17	10	11	773	858	0.90
dk27	9	9	131	146	0.90
dk48	15	17	1253	3687	0.34
f51m	8	8	2048	809	2.53
gary	15	11	3986	7522	0.53
in0	15	11	3513	4855	0.72
in1	16	17	9049	10209	0.89
in2	19	10	6071	8930	0.68
in3	35	29	16599	12567	1.32
in4	32	20	54601	45317	1.20
in5	24	14	14318	9558	1.50
in6	33	23	30057	11490	2.62
in7	27	10	1899	763	2.49
jbp	122	23	*	*44180	-
misg	56	23	16800	1063	15.80
mish	94	43	*	*5402	-
mlp4	8	8	3953	2644	1.50
opa	17	69	26848	23099	1.16
radd	8	5	586	415	1.41
rckl	32	7	2512	2229	1.13
rd53	5	3	38	57	0.67
rd73	7	3	452	590	0.77
risc	8	31	612	1042	0.59
root	8	5	1063	760	1.40
sqn	7	3	331	167	1.98
sqr6	6	12	1071	626	1.71
tial	25	8	359	359	1.00
vg2	25	8	25646	21014	1.22
wim	4	7	45	50	0.90
x1dn	27	6	27135	19064	1.42
x6dn	39	5	5700	5738	0.99
x9dn	27	7	38071	30440	1.25
z4	7	4	345	239	1.44
Z5xp1	7	10	1151	827	1.39
Z9sym	9	1	1710	729	2.35
Total			1575443	1073004	1.70

Time unit : 0.01 sec

2-2 $|sm(2) - sm(5)| = 0.84$, select $k = 2$
2-3 $sm(2) < sm(5)$, let $v_2 = 0$, the second output is set to be in true phase. Clear the contents in the fifth column.
3-1 Recalculate the summation of each row

sm(1)	sm(2)	sm(3)	sm(4)	sm(5)	sm(6)
*	*	2.0	*	*	1.16

3-2 $sm(3) > sm(6)$, let $v_3 = 1$, the third output is set to be in complementary phase.

With the selected phase assignment (001), the final PLA requires seven product terms.

4 Experimental results

The proposed algorithm has been implemented on Sun 4/260 workstation in C language. Espresso_MV (version 2.3) [4] is used as the minimisation algorithm. The PLAs described in Reference 3 are used for output phase optimisation, and the results are shown in Table 1. The fourth column shows the results obtained with Playground [6]. The fifth column is the optimisation results of Sasao's approach [5] implemented in Espresso_MV. The sixth column is the results of our algorithm. The last column shows some results of exhaustive searching. Among the 49 PLAs, both Playground and Sasao's approach fail to generate the solution for Mish. However, the proposed algorithm can generate a good result within a short time. In addition to this PLA, there are 11 PLAs that the proposed algorithm generates the best result as compared with other algorithms. Although there are some PLAs that the proposed algorithm does not work as well with, it still improves the performance. Compared with the results of exhaustive searching, the proposed algorithm obtains the same results for 26 PLAs. Considering the total number of product terms for the 49 PLAs. Sasao's approach generates 4603 product terms, Playground generates 4516 product terms and the proposed algorithm generates only 4460 product terms.

In Table 2, the speed of our algorithm is compared with that of Sasao's approach. Because the execution time of Playground [6] is slower as compared with the execution time for Sasao's approach, only the time for Sasao's approach and our approach are listed in the Table. The column with the heading 'Sasao' represents the time used for Sasao's approach, the column with the heading 'Proposed' represents the time spent with our algorithm, and the column with the heading 'Speedup' represents the value that Sasao's time divided by the proposed time. From the Table, one can find that the proposed algorithm is faster than Sasao's approach for many PLAs. The average speedup is about 1.71, that is about 41% of time can be saved with our approach. Since those PLAs that Sasao's approach fails to generate phase vector are not included in the Table, the average speedup is underestimated. From experimental results, the time needed for phase assignment procedure is about 13% of total time. To sum up, the proposed algorithm is superior to other algorithms both in speed and optimisation quality.

5 Conclusions

A new algorithm for output phase optimisation has been proposed and implemented. This algorithm first minimises the onset cover and offset cover individually. With the minimised covers, cubes in both covers are checked if they meet some properties. From the results of checking, the cost for each possible phase assignment is estimated. An output phase with minimum or near minimum estimated cost is chosen as the desired solution. This algorithm has been implemented on a Sun 4/260 workstation in C language. The experimental results demonstrate the excellent performance in speed and optimisation results. On the average, this algorithm can save 41% of execution time comparing with Sasao's approach. Besides this, some large PLAs which conventional algorithms fail to process within reasonable time are optimised with the proposed algorithm. With this algorithm not only is speed improved, but the total number of product terms for 49 PLAs is reduced as compared with other algorithms.

6 References

1 SANGIOVANNI-VINCENTELLI, A.L.: 'An overview of synthesis systems'. Proceedings of IEEE Conference on *Custom integrated circuits*, 1985, pp. 221–225
2 HONG, S.J., CAIN, R.G., and OSTAPKO, D.L.: 'MINI: A heuristic approach for logic minimization', *IBM J. Res. Develop.*, 1974, **18**, pp. 443–458
3 BRAYTON, R.K., HACHTEL, G., McMULLEN, C., and SANGIOVANNI-VINCENTELLI, A.L.: 'Logic minimization algorithms for VLSI synthesis' (Kluwer: Hinghan, MA, 1984)
4 RUDELL, R.L., and SANGIOVANNI-VINCENTELLI, A.L.: 'Multiple valued minimization for PLA optimization', *IEEE Trans.*, 1987, **CAD-6**, (5), pp. 727–750
5 SASAO, T.: 'Input variable assignment and output phase optimization of PLA's', *IEEE Trans.*, 1984, **C-33**, pp. 879–894
6 WEY, C.L., and CHANG, T.Y.: 'An efficient output phase assignment for PLA minimization', *IEEE Trans.*, 1990, **CAD-9**, (1), pp. 1–8