MINIMIZATION OF EXCLUSIVE SUMS OF MULTI-VALUED COMPLEX TERMS FOR LOGIC CELL ARRAYS

Ning Song and Marek Perkowski,

Portland State University, Dept. of Electr. Engn., Portland, Oregon 97207,
Tel: 503-725-5411, Fax: 503-725-4882, *mperkows@ee.pdx.edu*

Abstract— The paper proposes a new layout-driven multi-level logic factorization methodology for regular arrays of two-input cells, that can find practical applications in fine-grain FPGA design, standard cell, gate matrix layout and sub-micron technologies. A new factorization algorithm for AND/OR/EXOR logic with multi-valued literals is introduced, that has application to minimization of Logic Cell Arrays, and improves on previous results [1, 2]. It is shown that an extended cube representation and efficient minimization rules can be used, that generalize the ESOP minimization approach from [3]. Results of program MINICT demonstrate big area savings for several functions.

1. Introduction

A new methodology for designing *Logic Cell Arrays* (LCA) has been introduced in [1, 2]. LCA is a rectangular array of AND, OR and XOR cells with negated inputs and outputs, and vertical and horizontal buses, similar to Fine Grain Field Programmable Gate Arrays. Such array realizes an *Exclusive Sum of Complex Terms* (ESCT), where complex terms are cascade combinations of two-input AND, OR and XOR gates with possible negated inputs, and assuming the same order of input variables in them. Thus, LCA is a generalization of an EXOR PLA. When input decoders to the PLA are also available and decoders' outputs are represented as *multi-valued literals* (mv literals, for short), the problem of LCA area minimization becomes that of multi-valued ESCT minimization, presented here. Our approach introduces a new logic representation for factorization of multi-level functions realized in a regular form.

A. An Example of LCA Implementation

The following example of a two bit adder illustrates the procedure of LCA minimization. This function has 4 inputs and 3 outputs and has been minimized as an ESOP of 8 product terms:

$f_0 = ac \oplus \bar{a}bd \oplus b\bar{c}d$, $f_1 = b \oplus d$, $f_2 = c \oplus bd \oplus a$

Thus the initial solution requires 8 rows and 7 columns. Each product term is mapped into one row. There are 4 columns needed for inputs and 3 columns needed for outputs. The order of input variables is (a, b, c, d). The initial implementation is shown in Figure 1. Since we assume that there are two buses in each column, only 6 columns are needed in this implementation.

By setting the order of the input variables as *(b, d, a, c)* the two product terms in f_1 and the three product terms in f_2 can be combined to one complex term respectively. Three product terms in f_0 can be factorized to two complex terms. The result from the factorization is:

$f_0 = (bd + a)c + bda$, which requires two rows;

$f_1 = b \oplus d$, which requires one row;

$f_2 = bd \oplus a \oplus c$, which requires one row.

The implementation is shown in Figure 2, which takes 4 rows and 6 columns. After the output folding, the final result is shown in Figure 3. In this case, horizontal buses are used for outputs, and the number of output columns is reduced to 0.

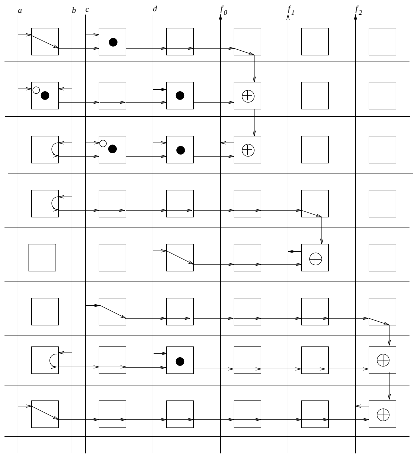

Figure 1: Initial implementation of the two-bit adder

Definition 1.1 *A* complex term *is a string of literals connected by a set of Boolean operators, and no literal can appear in the string more than once. The operators in the complex term can be any combinations of AND, OR, XOR, NAND, NOR, XNOR (denoted by* $\cdot$, $+$, $\oplus$, $\bar{\cdot}$, $\overline{+}$ *and* $\overline{\oplus}$ *respectively). All operators have the same priority. All operations must be performed in a sequence from* *left to right.*

Example 1.1 *Each of the following rows represents a complex term:* $(a\bar{b})+c$, $(a+b)c$, $(a\oplus b)+\bar{c}$, $((c\bar{b})+a)\oplus d$.

Example 1.2 $((ab)+\bar{b})c$ *is not a complex term, because the variable* b *appears twice.*

Example 1.3 $a+(b\bar{c})+d$ *is not a complex term because the operations are not in a sequence from left to right. However, if the order of variables is changed to* b, c, a, d, *then* $(b\bar{c})+a+d$ *becomes a complex term.*

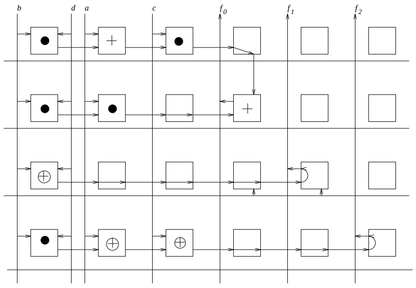

Figure 2: Implementation of the two-bit adder after row optimization

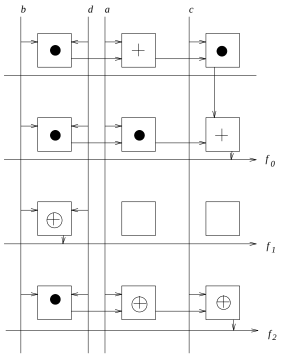

Figure 3: Final implementation of the two-bit adder

Definition 1.2 *An* multi-level cube *(*m-cube *for short) is a cube notation to present complex terms. Given a complex term, each literal of the complex term as well as each operator of the complex term is represented by a vector as shown below.*

$$c_1^0 c_1^1 ... c_1^{(p_1-1)} - d_2^0 d_2^1 ... d_2^{(m-1)} c_2^0 c_2^1 ... c_2^{(p_2-1)} - ... - d_n^0 d_n^1 ... d_n^{(m-1)} - c_n^0 c_n^1 ... c_n^{(p_n-1)}$$

where $c_i^0 c_i^1 ... c_i^{(p_i-1)}$ *is a* p_i *bit vector representing an* p_i*-valued literal, and* $d_i^0 d_i^1 ... d_i^{(m-1)}$ *is a* m *bit vector representing an operator.*

B. Basic Definitions.

The operators can be encoded in many different ways. One encoding scheme, which is referred to as *standard cording*, is: 00 - not used, 01 - AND, 10 - OR, 11 - XOR. In this coding scheme, Assume three types of operators are assumed to be used in a complex term: AND, OR and XOR. So two bits are needed to store each operator.

Example 1.4 *Given a complex term* $T = X_1^{\{001\}} \cdot X_2^{\{110\}} + X_3^{\{101\}} \oplus X_4^{\{011\}}$. *By using the standard coding, the complex term is presented by the m-cube as follows.*

$$\begin{array}{ccccccc} X_1 & \cdot & X_2 & + & X_3 & \oplus & X_4 \\ 001 & 01 & 110 & 10 & 101 & 11 & 011. \end{array}$$

Definition 1.3 *The* negation *of a complex term is assigned the value 1 if the complex term has the value 0; and is assigned the value 0 if the complex term has the value 1;*

Definition 1.4 *The operators in the complex terms are referred to as* literal operators. *The operators applied on a pair of complex terms are referred to as* term operators.

All the term operators and literal operators are listed as follows:

	AND	OR	XOR	NAND	NOR	XNOR
Term Term Operator	$\wedge$	$\vee$	$\uplus$	$\overline{\wedge}$	$\overline{\vee}$	$\overline{\uplus}$
Term Literal Operator	$\cdot$	$+$	$\oplus$	$\bar{\cdot}$	$\overline{+}$	$\overline{\oplus}$

Note, that each literal operator has its counterpart of term operator, and they have the same logic meaning. For instance, $\wedge = \cdot$. The difference is their priority. The literal operators are evaluated before the term operators.

Example 1.5 $a+b\cdot c\oplus d \vee \bar{a}\cdot\bar{b}+d \uplus \bar{\oplus} c+d = [(((a+b)\cdot c)\oplus d) \vee ((\bar{a}\cdot\bar{b})+d)] \uplus ((\bar{\oplus} c)+d)$.

Definition 1.5 *The* tail literal of a complex term *(*tail literal *for short), is the last literal of the complex term. The* tail operator of a complex term *(*tail operator *for short), is the last operator of the complex term. The tail literal and the tail operator together is referred to as the* tail of the complex term *(*tail *for short). The* head of a complex term, *(*head term *or* head *for short), are all the literals and operators of the complex term except its tail.*

Example 1.6 $a\cdot b+c\oplus d\cdot\bar{e}$ *is a complex term. The tail literal is* $\bar{e}$, *the tail operator is AND. The tail is* $\cdot\bar{e}$ *and* $a\cdot b+c\oplus d$.

When a complex term has only one literal, *e.g.* a, the literal is referred to as the tail. In this case, the complex term has no head.

Definition 1.6 *If the number of non-universal literals in a complex term is 1, the complex term is referred to as a* single literal complex term.

Definition 1.7 *A subterm of a complex term is defined as* **(1)** *the head of the complex term; or* **(2)** *the head of a subterm of the complex term.*

Example 1.7 *In example 1.6, the following are subterms:* $a \cdot b + c \oplus d$, $a \cdot b + c$, $a \cdot b$, a.

Please note that b is not a subterm of a complex term $a \cdot b + c$, because it is not the head of the complex term nor the head of any subterms of the complex term. In sequel, P, Q, or P_1, P_2 will be used to denote complex terms. P^n will be used to denote the complex term P has n input variables. Some of the literals in P may be universal, denoted by I. Consequently P^i will be used to denote a subterm of the complex term P^n, $(i < n)$.

Example 1.8 *Given a complex term* $P = X_1^{S_1} \odot X_2^{S_2} \odot \cdots \odot X_n^{S_n}$. *It can be presented by its subterms as*

$$\begin{aligned} P &= P^{n-1} \odot X_n^{S_n} \\ &= P^{n-2} \odot X_{n-1}^{S_{n-1}} \odot X_n^{S_n} \\ &\vdots \\ &= P^1 \odot X_2^{S_2} \odot \cdots \odot X_n^{S_n}. \end{aligned}$$

Since programmable inverters at each cell input are assumed to be available, not only $P(i-1) \odot \overline{X}_i^{S_i}$ can be implemented, but also $\overline{P(i-1) \odot \overline{X}_i^{S_i}}$. In other words, the head of the complex term can be negated or non-negated.

Example 1.9 $P = (a \cdot b) + \bar{c}$ *is a complex term,* $Q = \overline{a \cdot b} + \bar{c}$ *is also a complex term.*

Lemma 1.1 *All the Boolean properties, defined on two complex terms, can be applied on the head and the tail of a complex term.*

Definition 1.8 *A* positive complex term *is a complex term without NAND, NOR and XNOR operators. An ESCT contains only positive terms are referred to as in* positive term form*.*

By the above definition, no literal operators are negated in a positive complex term. Please note that the negations on input variables are allowed.

Example 1.10 $a \cdot \bar{b}$ *is a positive complex term.* $a \bar{\cdot} b$ *is not a positive complex term, because there is a NAND operator in the term.*

Definition 1.9 *An* or-free complex term *is a complex term without OR and NOR operators. An ESCT contains only or-free complex terms are referred to as in* or-free form*.*

In an or-free complex term, its literal operators may be any combinations of AND, NAND and XOR operators. XNOR operators will be normalized to XOR, as discussed in more detail in [4].

Example 1.11 $a \oplus b \bar{\cdot} c$ *is an or-free complex term.* $a + b \bar{\cdot} c$ *is not a or-free complex term, because it contains an OR operator.*

C. The ESCT Minimization Problem.

The starting point to ESCT minimization is a minimized *Exclusive Sum of Products* (ESOP) expression. The main idea of the ESOP minimization, as implemented in programs EXORCISM-MV-2 and EXORCISM-MV-3, [3, 4], is to *link* (reshape) a pair of *product terms.* This is done using *exorlink operations* [3, 4]. The product terms, after reshaping, may be able to merge with other product terms. Thus the total number of product terms in an ESOP is minimized. The same idea is used in the ESCT minimization proposed below. The key operation in the ESCT minimization is to link (reshape) a pair of *complex terms.* We found that the *complex term linking* could be done is a way similar to the product term (exor)linking. As shown in [4], any two product terms in an ESOP can be linked and can generate a number of resultant product terms. The number of resultant product terms depends on the *distance* of two product terms. We discovered that this property exists in the ESCT as well. We first show in section 2 that any two complex terms in an ESCT can be linked just like the exorlink of two product terms in an ESOP.

2. Multiple Level EXOR Linking

A. Linking Two Complex Terms

Given two complex terms $P_1 = P_1^{n-1} \cdot X_n^{S_n}$ and $P_2 = P_2^{n-1} \cdot X_n^{R_n}$, the following properties hold.

Property 2.1. $(P_1^{n-1} \cdot X_n^{S_n}) \uplus (P_2^{n-1} \cdot X_n^{R_n})$
$= [(P_1^{n-1} \uplus P_2^{n-1}) \cdot X_n^{S_n}] \uplus (P_2^{n-1} \cdot X_n^{S_n \uplus R_n})$
$= [(P_1^{n-1} \uplus P_2^{n-1}) \cdot X_n^{R_n}] \uplus (P_1^{n-1} \cdot X_n^{S_n \uplus R_n})$

Property 2.1 shows that two complex terms can be linked just like two product terms.

Property 2.2. $(P_1^{n-1} \cdot X_n^{S_n}) \overline{\uplus} (P_2^{n-1} \cdot X_n^{R_n})$
$= [(P_1^{n-1} \uplus P_2^{n-1}) \cdot X_n^{S_n}] \overline{\uplus} (P_2^{n-1} \cdot X_n^{S_n \uplus R_n})$
$= [(P_1^{n-1} \uplus P_2^{n-1}) \cdot X_n^{R_n}] \overline{\uplus} (P_1^{n-1} \cdot X_n^{S_n \uplus R_n})$

Property 2.2 shows that if the term operator is XNOR, we can link the two complex terms just like in the Property 2.1, and keep the term operator as XNOR. This property has no direct application on two complex terms, because in ESCT, the term operators are always XOR gates. However, when we apply the linking on two subterms, the property is useful.

Theorem 2.1 *Any two complex terms in a ESCT can be linked.*

The following example shows the linking of two complex terms with different combinations of tail operators.

Example 2.1 *Given two complex terms* $P_1 = P_1^{n-1} + X_n^{S_n}$ *and* $P_2 = P_2^{n-1} \oplus X_n^{R_n}$. $(P_1^{n-1} + X_n^{S_n}) \uplus (P_2^{n-1} \oplus X_n^{R_n})$ $= \overline{\overline{\overline{P_1^{n-1}} \cdot \overline{X_n^{S_n}}}} \uplus P_2^{n-1} \uplus X_n^{R_n} = \overline{P_1^{n-1}} \cdot \overline{X_n^{S_n}} \uplus P_2^{n-1} \uplus \overline{X_n^{R_n}}$ $= [(\overline{P_1^{n-1}} \uplus P_2^{n-1}) \cdot I] \uplus (\overline{P_1^{n-1}} \cdot X_n^{S_n \uplus I}) \uplus \overline{X_n^{R_n}} = [(\overline{P_1^{n-1}} \uplus P_2^{n-1})] \uplus (\overline{P_1^{n-1}} \cdot \overline{X_n^{S_n}}) \uplus \overline{X_n^{R_n}} = (\overline{P_1^{n-1}} \uplus P_2^{n-1}) \uplus \overline{X_n^{R_n}} \uplus (\overline{P_1^{n-1}} \cdot \overline{X_n^{S_n}}) = [(\overline{P_1^{n-1}} \uplus P_2^{n-1}) \oplus \overline{X_n^{R_n}}] \uplus (\overline{P_1^{n-1}} \cdot \overline{X_n^{S_n}})$.

In general case, Property 2.1 generates three complex terms: $(P_1^{n-1} \uplus P_2^{n-1}) \cdot X_n^{S_n}$ are a pair of two complex terms, and $P_2^{n-1} \cdot X_n^{S_n \uplus R_n}$ is a single complex term. If we continue to apply this property to the subterms (P_1^{n-1} and P_2^{n-1}), we may generate three subterms from two subterms. This process could continue until the subterms are single literal terms.

Example 2.2 *Given are two complex terms* $P_1 = a + b \oplus c \cdot d$ *and* $P_2 = a \oplus b \cdot c + d$. *The linking of* P_1 *and* P_2 *consists of the following steps:*
(1) *Linking* P_1 *and* P_2, *the heads are* $P_1^3 = a + b \oplus c$ *and* $P_2^3 = a \oplus b \cdot c$, *the tails are* $\cdot d$ *and* $+d$. $P_1 \uplus P_2 = (P_1^3 \cdot d) \uplus (P_2^3 + d) = (P_1^3 \cdot d) \overline{\uplus} (\overline{P_2^3} \cdot \overline{d}) \overset{(by\ Property 2.2)}{=\!=\!=\!=\!=} [(P_1^3 \uplus \overline{P_2^3}) \cdot d] \overline{\uplus} [P_2^3 \cdot (d \oplus \overline{d})] = [(P_1^3 \uplus \overline{P_2^3}) \cdot d] \overline{\uplus} [P_2^3 \cdot I] = [(P_1^3 \uplus \overline{P_2^3}) \cdot d] \overline{\uplus} P_2^3 = [(P_1^3 \overline{\uplus}\, P_2^3) \cdot d] \uplus \overline{P_2^3}$. $[(P_1^3 \overline{\uplus}\, P_2^3) \cdot d]$ *is a pair of complex terms which will be linked in the next step.* $\overline{P_2^3}$ *is a single complex term.* $\overline{P_2^3} = \overline{a \oplus b \cdot c} = \bar{a} \oplus b + \bar{c}$. *There are three resultant complex terms after step 1:* $P_1^3 \cdot d$, $P_2^3 \cdot d$, *and* $\bar{a} \oplus b + \bar{c}$.
(2) *Linking two subterms* $P_1^3 = a + b \oplus c$ *and* $P_2^3 = a \oplus b \cdot c$, *the heads are* $P_1^2 = a + b$ *and* $P_2^2 = a \oplus b$, *the tails are* $\oplus c$ *and* $\cdot c$. $P_1^3 \overline{\uplus}\, P_2^3 = (P_1^2 \oplus c) \overline{\uplus} (P_2^2 \cdot c) = P_1^2 \uplus c \overline{\uplus} (P_2^2 \cdot c) = (P_1^2 \cdot I) \overline{\uplus} (P_2^2 \cdot c) \uplus c = \overset{(by\ Property 2.2)}{=\!=\!=\!=\!=} = [(P_1^2 \uplus P_2^2) \cdot c] \overline{\uplus} [P_1^2 \cdot (I \uplus c)] \uplus c = [(P_1^2 \overline{\uplus}\, P_2^2) \cdot c] \uplus c \overline{\uplus} (P_1^2 \cdot \bar{c}) = \overline{(P_1^2 \uplus P_2^2)} \cdot c \overline{\uplus} (P_1^2 \cdot \bar{c}) = (P_1^2 \overline{\uplus}\, P_2^2) \cdot c \uplus (\overline{P_1^2} + c)$. $(P_1^2 \overline{\uplus}\, P_2^2) \cdot c$ *will be linked at the next step.* $(\overline{P_1^2} + c)$ *is a single complex term.* $(\overline{P_1^2} + c) = \overline{a + b} + c = \bar{a} \cdot \bar{b} + c$. *There are four resultant complex terms after step 2:* $P_1^2 \cdot c \cdot d$, $P_2^2 \cdot c \cdot d$, $\bar{a} \cdot \bar{b} + c \cdot d$, $\bar{a} \oplus b + \bar{c}$. *Note, that the last resultant complex term,* $\bar{a} \oplus b + \bar{c}$, *is generated by the previous step. The other resultant complex terms all have a tail* $\cdot d$, *also found in the previous step.*
(3) *Linking two subterms* $P_1^2 = a + b$ *and* $P_2^2 = a \oplus b$, *the heads are* $P_1^1 = a$ *and* $P_2^1 = a$, *the tails are* $+b$ *and* $\oplus b$. $P_1^2 \overline{\uplus} P_2^2 = (P_1^1 + b) \overline{\uplus} (P_2^1 \oplus b) = (\overline{P_1^1} \cdot \bar{b}) \uplus P_2^1 \uplus b = (\overline{P_1^1} \uplus P_2^1) \uplus [\overline{P_1^1} \cdot (I \uplus \bar{b})] \uplus b = (\overline{P_1^1} \uplus P_2^1) \uplus (\overline{P_1^1} \cdot b) \uplus b = (\overline{P_1^1} \uplus P_2^1) \uplus b \uplus (\overline{P_1^1} \cdot b) = [(\overline{P_1^1} \uplus P_2^1) \oplus b] \uplus (\bar{a} \cdot b)$. *There are five resultant complex terms after step 3:* $\overline{P_1^1} \oplus b \cdot c \cdot d$, $P_2 \oplus b \cdot c \cdot d$, $\bar{a} \cdot b \cdot c \cdot d$, $\bar{a} \cdot \bar{b} + c \cdot d$, $\bar{a} \oplus b + \bar{c}$.
(4) *In the last step, both* P_1^1 *and* P_2^1 *are single literals.* $P_1^1 = a$. $P_2^1 = a$. $\overline{P_1^1} \uplus P_2^1 = \bar{a} \uplus a = I$. $(\overline{P_1^1} \uplus P_2^1) \oplus b = I \oplus b = \bar{b}$. *The final results are the following four complex terms:* $\bar{b} \cdot c \cdot d$, $\bar{a} \cdot b \cdot c \cdot d$, $\bar{a} \cdot \bar{b} + c \cdot d$, $\bar{a} \oplus b + \bar{c}$.

Definition 2.1 *If a head of a complex term is an universal subterm, then the head is referred to as an* universal head. *If a tail literal is universal, then the tail is referred to as an* universal tail.

An universal head with i literals will be denoted as I^i. An universal tail literal will be denoted as I. We found that the following conditions may reduce the number of resultant complex terms.

1. identical literals or identical literal operators,
2. complement heads ($P_1 = \overline{P_2}$) or complement tails ($X_n^{S_n} = \overline{X_n^{R_n}}$),
3. universal literals,
4. one tail operator is AND or OR, the other tail operator is XOR.

Comparing with the product term linking, there is only one condition, identical literals, that has impact on the number of resultant product terms. The principle is: we try to find those pairs of complex terms which generate a small number of resultant complex terms by linking.

B. Different Linking Cases

The previous subsection shows that linking of two complex terms can be done step-by-step linking the heads and tails of the complex terms and their subterms. In this section, linking of heads and tails of two complex terms is discussed. In general the two complex terms can be written in the form: $P_1 = P_1^{n-1} \odot_1 X_n^{S_n}$, $P_2 = P_2^{n-1} \odot_2 X_n^{R_n}$ where $\odot_1$ and $\odot_2$ are two *tail operators* in P_1 and P_2 respectively. There are four cases to be investigated for heads:
[1] $P_1^{n-1} = P_2^{n-1}$;
[2] $P_1^{n-1} = \overline{P_2^{n-1}}$;
[3] $P_1^{n-1} \neq P_2^{n-1}$, $P_1^{n-1} = I^{n-1}$;
[4] $P_1^{n-1} \neq P_2^{n-1}$, $P_1^{n-1} \neq I^{n-1}$, $P_2^{n-1} \neq I^{n-1}$;
Note that case 1 includes the case that both heads are universal. The case 2 is that one of the heads is the negation of the other. Case 3 is the case that one of the heads is universal. Since the commutative law holds, when either one of the heads is universal, belongs to this case. The last case is that the two heads are different and none of them is universal. Thus all the different cases are included in the above four cases.
Similarly, there are four different cases for tail literals:
[1] $X_n^{S_n} = X_n^{R_n}$;
[2] $X_n^{S_n} = \overline{X_n^{R_n}}$;
[3] $X_n^{S_n} \neq X_n^{R_n}$, $X_n^{R_n} = I$;
[4] $X_n^{S_n} \neq X_n^{R_n}$, $X_n^{S_n} \neq I$, $X_n^{R_n} \neq I$.
Please note that in case 3, one of the tail literals is universal. If the head is also in case 3, the universal head and the universal tail literal are in different complex terms. In other words, the case is either P_1^{n-1} and $X_n^{R_n}$ are universal, or P_2^{n-1} and $X_n^{S_n}$ are universal. If both P_1^{n-1} and $X_n^{S_n}$ are universal, or both P_2^{n-1} and $X_n^{R_n}$ are universal, the case should be taken care of at a previous stage by evaluating $P_1^n \odot X_{n+1}^{S_{n+1}}$ and $P_2^n \odot X_{n+1}^{R_{n+1}}$. So at the current stage, if both the head and the tail are in case 3, we only need to consider the case that the universal head and the universal tail are in different complex terms.
There are six cases of positive literal operators: **(1)** both operators are AND; **(2)** both operators are OR; **(3)** both operators are XOR; **(4)** one operator is AND and the other is OR; **(5)** one operator is AND and the other is XOR; **(6)** one operator is OR and the other is XOR. There are also the cases that the literal operators are **negative**.

Table 1: Linking Cases for the Positive Term Form

	$X_{i+1}^{S_{i+1}} = X_{i+1}^{R_{i+1}}$	$X_{i+1}^{S_{i+1}} = I^i$	$X_{i+1}^{S_{i+1}} = \overline{X_{i+1}^{R_{i+1}}}$	$X_{i+1}^{S_{i+1}} \neq X_{i+1}^{R_{i+1}}$
$P_1^i = P_2^i$	$(\cdot,\cdot)$ $(+,+)$ $(\oplus,\oplus)$ $(\cdot,+)$ $(\cdot,\oplus)$ $(+,\oplus)$	$(\cdot,\cdot)$ $(\cdot,+)$ $(\cdot,\oplus)$	$(\cdot,\cdot)$ $(+,+)$ $(\oplus,\oplus)$ $(\cdot,+)$ $(\cdot,\oplus)$ $(+,\oplus)$	$(\cdot,\cdot)$ $(+,+)$ $(\oplus,\oplus)$ $(\cdot,+)$ $(\cdot,\oplus)$ $(+,\oplus)$
$P_2^i = I$	$(\cdot,\cdot)$ $(\cdot,+)$ $(\cdot,\oplus)$	$(\cdot,\cdot)$	$(\cdot,\cdot)$ $(\cdot,+)$ $(\cdot,\oplus)$	$(\cdot,\cdot)$ $(\cdot,+)$ $(\cdot,\oplus)$
$P_1^i = \overline{P_1^i}$	$(\cdot,\cdot)$ $(+,+)$ $(\oplus,\oplus)$ $(\cdot,+)$ $(\cdot,\oplus)$ $(+,\oplus)$	$(\cdot,\cdot)$ $(\cdot,+)$ $(\cdot,\oplus)$	$(\cdot,\cdot)$ $(+,+)$ $(\oplus,\oplus)$ $(\cdot,+)$ $(\cdot,\oplus)$ $(+,\oplus)$	$(\cdot,\cdot)$ $(+,+)$ $(\oplus,\oplus)$ $(\cdot,+)$ $(\cdot,\oplus)$ $(+,\oplus)$
$P_1^i \neq P_2^i$	$(\cdot,\cdot)$ $(+,+)$ $(\oplus,\oplus)$ $(\cdot,+)$ $(\cdot,\oplus)$ $(+,\oplus)$	$(\cdot,\cdot)$ $(\cdot,+)$ $(\cdot,\oplus)$	$(\cdot,\cdot)$ $(+,+)$ $(\oplus,\oplus)$ $(\cdot,+)$ $(\cdot,\oplus)$ $(+,\oplus)$	$(\cdot,\cdot)$ $(+,+)$ $(\oplus,\oplus)$ $(\cdot,+)$ $(\cdot,\oplus)$ $(+,\oplus)$

Table 2: Linking Cases for the Or-Free Form

	$X_{i+1}^{S_{i+1}} = X_{i+1}^{R_{i+1}}$	$X_{i+1}^{S_{i+1}} = I^i$	$X_{i+1}^{S_{i+1}} = \overline{X_{i+1}^{R_{i+1}}}$	$X_{i+1}^{S_{i+1}} \neq X_{i+1}^{R_{i+1}}$
$P_1^i = P_2^i$	$(\cdot,\cdot)$ $(\oplus,\oplus)$ $(\cdot,\oplus)$	$(\cdot,\cdot)$ $(\cdot,\oplus)$	$(\cdot,\cdot)$ $(\oplus,\oplus)$ $(\cdot,\oplus)$	$(\cdot,\cdot)$ $(\oplus,\oplus)$ $(\cdot,\oplus)$
$P_2^i = I$	$(\cdot,\cdot)$ $(\cdot,\oplus)$	$(\cdot,\cdot)$	$(\cdot,\cdot)$ $(\cdot,\oplus)$	$(\cdot,\cdot)$ $(\cdot,\oplus)$
$P_1^i = \overline{P_1^i}$	$(\cdot,\cdot)$ $(\oplus,\oplus)$ $(\cdot,\oplus)$	$(\cdot,\cdot)$ $(\cdot,\oplus)$	$(\cdot,\cdot)$ $(\oplus,\oplus)$ $(\cdot,\oplus)$	$(\cdot,\cdot)$ $(\oplus,\oplus)$ $(\cdot,\oplus)$
$P_1^i \neq P_2^i$	$(\cdot,\cdot)$ $(\oplus,\oplus)$ $(\cdot,\oplus)$	$(\cdot,\cdot)$ $(\cdot,\oplus)$	$(\cdot,\cdot)$ $(\oplus,\oplus)$ $(\cdot,\oplus)$	$(\cdot,\cdot)$ $(\oplus,\oplus)$ $(\cdot,\oplus)$

However, all the negative operators can be converted to the positive operators without increasing the number of complex terms. So only the positive operators are considered here. There are totally 4 (case on heads) $\times$ 4 (case on tail literals) $\times$ 6 (cases on tail operators) = 96 different cases.

In [4], the concept of simplified complex terms is introduced. In a simplified complex term, some of the above conditions would not occur. For instance, $P^{n-1} \oplus I$ is not a simplified complex term. Table 1 lists all the different linking cases. Since the simplified complex terms are assumed, the tail operators adjacent to universal literals are AND only, there are 23 cases which do not exist. So there are 73 cases listed in Table 1.

In Table 1, the top left block shows the cases that the two complex terms have the identical heads and identical tails. There are six cases in this block, which are all combinations of tail operators. In the first row, the second block from the left has three cases: **(1)** both operators are AND, **(2)** one operator is AND and the other is OR, **(3)** one operator is AND and the other is XOR. This block includes the cases that the two complex terms have the identical heads but different tail literals, and one of the tail literal is universal. According to the simplification rules from [4], if the tail literal is universal, the tail operator is AND. So only three cases instead of six cases are in this block.

The above discussion is based on the *positive term form.* Comparing with the *or-free form*, we found that the latter one has two advantages: **(1)** If the or-free form is used, the number of linking cases is less than in the positive term form. This is because the OR operators are eliminated from the ESCT. The combinations of literal operators are reduced. **(2)** $P_1^i = \overline{P_2^i}$ is one of the conditions to be checked before the linking. Checking this condition is more convenient by using the or-free form than the positive term form.

There are 40 cases in Table 2. This number is significantly smaller than in the positive form, which is 73. All these cases were proved in [4].

Two Complex Terms with Identical Tails We consider here the case that the tails of the two complex terms are the same, and the two heads are different. In this case, the tail operators can be AND, OR or XOR. The linking of such two complex terms are shown by the following three equations:

$$(P_1^{n-1} \cdot X_n^{S_n}) \uplus (P_2^{n-1} \cdot X_n^{S_n}) = (P_1^{n-1} \uplus P_2^{n-1}) \cdot X_n^{S_n} \quad (1)$$

$$(P_1^{n-1} + X_n^{S_n}) \uplus (P_2^{n-1} + X_n^{S_n}) = (P_1^{n-1} \uplus P_2^{n-1}) \cdot \overline{X_n^{S_n}} \quad (2)$$

$$(P_1^{n-1} \oplus X_n^{S_n}) \uplus (P_2^{n-1} \oplus X_n^{S_n}) = P_1^{n-1} \uplus P_2^{n-1} \quad (3)$$

Above three equations show that if the tails of two positive terms are the same, then the tail can be extracted out. To extract an identical tail, the following operations are performed: **(1)** if the tail operator is AND, the tail remains the same; **(2)** if the tail operator is OR, the tail is negated; **(3)** if the tail operator is XOR, the tail becomes an universal tail.

As proved in [4], the above three equations can be then applied on the subterms if the tails of the two subterms are also the same. Note that if the or-free form is used, then the OR operators have been converted to AND operators and Equation (2) above : B is not needed. However, converting to or-free form does not change the linking results.

3. Outline of MINICT program

Since the linking rules for complex terms are much more complicated than the linking rules for product terms, all the different linking cases are discussed in [4], as well as several special cases. The new cube operation, *m-link,* which is an extension of exorlink, has been next introduced. Just like for the exorlink, we proved in [4] that the m-link operation can be applied on any two complex terms in a ESCT, and the number of resultant complex terms depends on the distance of two complex terms. Then the distance of two complex terms is defined with a thorough analysis of all different linking cases, the approach very similar in principle to one from [5], and being an extension from [3]. Finally, program MINICT, the ESCT minimizer has been created based on look-ahead search using m-link operations of different distances, the ideas generalizing those presented in [5].

4. Experimental Results of MINICT

The presented algorithm has been implemented in program MINICT. The experimental results are shown in Table 3. The inputs of the program are minimized ESOPs. The outputs are ESCTs with minimized number of complex terms and minimized number of output columns. In Table 3, the columns In and Out are number of input variables and number of output variables, respectively. C_{T0} is the number of product terms in the ESOPs. C_{in} is the initial cost. According to the cost function defined in Section 5 the initial cost; $C_{in} = C_{T0} \times (In + Out)$. In Table 3, the column C_T is the number of complex terms after complex terms minimization. The column C_{CO} is the number of output columns after column folding [1,2,4]. Column C is the cost of minimized ESCTs. $C = C_T \times (In + C_{CO})$. Column C/C_{in} gives the ratio of the costs after the minimization and before the minimization. $Time$ is cpu seconds on a Pentium 133 MHz PC. For instance, the first test case, 5xp1, has 7 inputs, 10 outputs, and 32 product terms. The initial cost is $32 \times (7 + 10) = 544$. It has been minimized to 8 complex terms and 7 output columns. The cost is $8 \times (7 + 7) = 112$. The run time is 0.10 seconds. The cost ratio is $112 / 544 = 0.21$.

Table 3 shows that for some test cases, the costs are significantly reduced. While for others, there is little or no reductions. Our suggestion is that we could try also SCT (sum of complex terms) minimizations. If both SCT minimizations and ESCT minimizations are performed and the better results between the two are selected, the overall results could be improved.

Observe also, that MINICT starts from an already optimized ESOP form, so comparing the areas and delay times of initial SOP or ESOP PLAs and the final LCAs produced, would be even much more advantageous for the Exorcism-mv/MINICT/folding combination used by us.

Table 3: MINICT on MCNC Test Cases

	In	Out	C_{T0}	C_{in}	C_T	C_{CO}	C	C/C_{in}	Time
5xp1	7	10	32	544	8	7	112	0.21	0.10
9sym	9	1	51	510	45	1	450	0.88	0.86
adr4	8	5	31	403	16	1	144	0.36	0.06
b12	15	9	28	672	27	3	486	0.72	374.42
bw	5	28	22	726	22	19	528	0.73	4.06
clip	9	5	63	882	62	5	868	0.98	6.79
con1	7	2	9	81	8	1	64	0.79	0.02
f51m	8	8	31	496	30	7	450	0.91	0.39
inc	7	9	26	416	25	9	400	0.96	1.10
log8	8	8	83	1328	82	8	1312	0.99	16.98
misex1	8	7	12	180	12	7	180	1	0.19
misex2	25	8	27	891	26	4	754	0.85	0.69
mlp4	8	8	60	960	57	5	741	0.77	2.82
nrm4	8	5	67	871	66	5	858	0.99	6.72
rd53	5	3	14	112	9	1	54	0.48	0.03
rd73	7	4	35	385	25	1	200	0.52	0.28
rd84	8	4	59	708	46	1	414	0.58	0.48
rdm8	8	8	31	496	29	7	435	0.88	0.43
rot8	8	5	35	455	34	5	442	0.97	0.95
sao2	10	4	28	392	28	4	392	1	0.60
squar5	5	8	19	247	17	5	170	0.69	0.05
t481	16	1	13	221	12	1	204	0.92	0.05
wgt8	8	4	56	672	36	2	360	0.54	0.34
xor5	5	1	5	30	1	1	6	0.2	0.01
Total	212	155	837	12678	723	110	10024	0.79	418.42

5. Conclusions

The main contribution of this paper is the proposition of a comprehensive logic design representation and corresponding minimization algorithms for both ASICs and FPGAs. While connection oriented synthesis tools are highly needed by both ASIC and FPGA designers, the current methodologies have been facing difficulties to meet the demand. Based on complex term theorems [4], we developed minimization algorithms for logic cell array optimization. It is for the first time that multi-level factorization method was given that applies to AND, OR and EXOR gates and addresses concurrently the issues of space, delay times and regularity. Although our theorems play a fundamental role to create linking subroutine for MINICT, they are of the value by themselves, and can be used for a more general tree factorization, and also extended to SCT factorization that starts from SOP rather than ESOP form. All presented ESCT methods have their counterparts in corresponding new SCT methods. Finally, some output functions can be realized by SCT and some by ESCT, which will further improve the results.

Our methodology provides a new concept and an alternative approach to meet the challenge. The proposed logic representation and optimization algebra has several important advantages over the existing logic representations and methodologies: **(1)** The logic representation and design implementation are consistent. **(2)** The stages of logic synthesis and physical design are effectively merged into a single stage. **(3)** The structure of the mapping solution is a regular rectangle. **(4)** Since the connections are mainly between neighbor cells, the wire delay is reduced comparing to other design methods. **(5)** Since the structure is regular, the creation of the high-performance tools will be significantly easier.

References

[1] A. Sarabi, N. Song, M. Chrzanowska-Jeske and M. Perkowski, "A Comprehensive Approach to Logic synthesis and Physical Design for two-dimensional Logic Arrays," *Proc. 31th ACM/IEEE Design Automation Conference*, pp. 321-326, June, 1994.

[2] N. Song, M. Perkowski, M. Chrzanowska-Jeske and A. Sarabi, "A New Design Methodology for Two-Dimensional Logic Arrays," *VLSI Design*, 1995, Vol. 3, Nos. 3-4, pp. 315-332.

[3] N. Song and M. Perkowski, "Minimization of Exclusive Sum of Products Expressions for Multiple-Valued Input Incompletely Specified Functions," *IEEE Trans. on CAD*, Vol. 15, no. 4, pp. 385-395, 1996.

[4] N. Song, "A New Design Methodology for Two-Dimensional Logic Cell Arrays," *Ph.D. Thesis*, November 14, 1997, PSU.

[5] N. Song, M. Perkowski, "A New Fast Approach to Approximate ESOP Minimization for Incompletely Specified Multi-Output Functions," *Proc. RM'97*, pp. 61 - 72.