# Introducing Read-Polarity-Once Functions

**Vinicius Callegaro, Renato P. Ribas, André I. Reis**
{vcallegaro, rpribas, andreis}@inf.ufrgs.br

**PPGC – UFRGS, Porto Alegre – RS, Brazil**

## Abstract

*Efficient exact factoring algorithms exist in the literature, but they are limited to read-once (RO) functions where each variable appears once in the final equation. These algorithms have two limitations: (1) they do not consider incompletely specified functions; and (2) they are not applicable to binate functions. To overcome the first limitation, we propose an algorithm that finds RO formulas for incompletely specified functions, whenever possible. To overcome the second limitation, we propose a domain transformation that splits existing binate variables into two independent unate variables. This domain transformation leads to incompletely specified functions, which can be efficiently factored with the proposed algorithm. The combination of the proposed factoring algorithm and the domain transformation gives exact results for a novel broader class of functions called read-polarity-once (RPO) functions, where each polarity (positive or negative) of a variable appears at most once in the factored form.*

## 1. Introduction

Factoring Boolean functions [1] is one of the basic operations in algorithmic logic synthesis [2]. Factoring is the process of deriving a parenthesized algebraic expression or factored form representing a given logic function, usually provided initially in a sum-of-products form (SOP) or product-of-sums (POS) form. For example, F=*a*c+b*c+c*d*e* can be factored into the logically equivalent form F=*c(a+b+de)*.

In general, a logic function will have many factored forms. The problem of factoring Boolean functions into shorter, more compact logically equivalent formulae is one of the basic operations in the early stages of algorithmic logic synthesis. In most design styles (like CMOS design) the implementation of a Boolean function corresponds directly to its factored form. Generating an optimum factored form (a shortest length expression) is an NP-hard problem, thus heuristic algorithms [1-5] have been developed in order to obtain good factored forms. Good heuristic algorithms include Xfactor [3-4], which provides good results but does not guarantee optimal results. Optimal results for general expressions are known for a long time [6], but the algorithms are too slow to be used in practice. Recently some exact results with better runtime properties have been proposed [7-8]. However, the runtimes are still not comparable to the runtime efficiency of heuristic algorithms like Xfactor [3-4].

Efficient exact algorithms exist [9-11], but they are limited to the class of Boolean functions known as read-once functions [12]. Read once functions can be written by using formulae where each variable appears at most once. Exact algorithms for read once functions have two limitations: (1) they do not consider incompletely specified functions; and (2) they are not applicable to binate functions. To overcome the first limitation, we propose an algorithm that finds RO formulas for incompletely specified functions, whenever possible. To overcome the second limitation, we propose a domain transformation that splits existing binate variables into two independent unate variables. This domain transformation leads to incompletely specified functions, which can be efficiently factored with the proposed algorithm. The combination of the proposed factoring algorithm and the domain transformation gives exact results for a novel broader class of functions called read-polarity-once (RPO) functions, where each polarity (positive or negative) of a variable appears at most once in the factored form.

This paper is organized as follows. Section 2 presents basic concepts for the understanding of the paper. Section 3 presents an algorithm for factoring Incompletely Specified Read-Once (ISRO) functions. Section 4 presents the proposed domain transformation to split existing binate variables into two independent unate variables. The complete algorithm to perform factoring of Read-Polarity-Once (RPO) functions is presented in Section 5. Results are presented in Section 6, followed by conclusions in Section 7.

## 2. Basic concepts

The algorithms we propose here are strongly based on the concept of cofactor. The operation of cofactoring a logic function can be performed in several ways, and comparison between cofactors of a same variable can be used to determine the polarity of variables in the function. Consider the three example functions shown in Table 1. Function f could be written in hexadecimal code as f=(80)hexa. Similarly, g and h

could be written as g=(01)hexa and h=(96)hexa. The positive cofactor of function f with respect to variable a is obtained by making a=1 in f, which results f(a=1)=88hexa. The negative cofactor of function f with respect to variable a is obtained by making a=0 in f, which results f(a=0)=00hexa. Notice that when functions are represented by strings of bits, cofactoring can be done by performing masking and rotations in the strings of bits. This is common practice for current logic synthesis tools [13] and details can be found in [14].

Tab. I - Three example functions

| *a* | *b* | *c* | *f=a*b*c* | *g=!(a+b+c)* | *h=a^b^c* |
|---|---|---|---|---|---|
| *0* | *0* | *0* | *0* | *1* | *0* |
| *0* | *0* | *1* | *0* | *0* | *1* |
| *0* | *1* | *0* | *0* | *0* | *1* |
| *0* | *1* | *1* | *0* | *0* | *0* |
| *1* | *0* | *0* | *0* | *0* | *1* |
| *1* | *0* | *1* | *0* | *0* | *0* |
| *1* | *1* | *0* | *0* | *0* | *0* |
| *1* | *1* | *1* | *1* | *0* | *1* |

Consider now the polarity of variables in the functions f, g and h. A variable can have one out of four distinct possible polarities in a function: (1) don't care, (2) positive unate, (3) negative unate and binate. Consider the cofactors of variable a shown in Table 2. For function f, variable *a* is a positive unate variable, as the bitwise or of the positive and negative cofactors is equal to the positive cofactor. For function g, variable *a* is a negative unate variable, as the bitwise or of the positive and negative cofactors is equal to the negative cofactor. For function h, variable *a* is a binate variable, as both cofactors are not equal and they are distinct from the bitwise (BW) or of themselves.

Tab. II - Positive and negative cofactors of variable a for functions f, g and h

| FUNCTION | COFACTOR A=1 | COFACTOR A=0 | BW OR OF COFACTORS | BW AND OF COFACTORS |
|---|---|---|---|---|
| F | 88 | 00 | 88 | 00 |
| G | 00 | 11 | 11 | 00 |
| H | 99 | 66 | FF | 00 |

Computing the polarity of the variables is important because all read once functions are unate functions. This means that in a read once-function every variable has to be either positive unate or negative unate. This is expressed by lemma 1. However, not every unate function is a read once function. This is expressed by lemma 2. The work of Elbassioni et al [15] investigates how many times a variable has to be read in unate functions. Lemma 3 states an optimality observation for functions containing binate variables.

**Lemma 1:** every read-once function is unate in all its variables.

**Lemma 2:** not every unate function is a read-once function.

**Lemma 3:** an equation is in the minimum literal form if each variable is read at most once for each polarity of the variable, considering a binate variable as having both positive and negative polarities.

# 3. Factoring RO functions

Efficient algorithms exist to perform factoring of read once formulas. Most of them readily abandon functions containing binate variables, as a read-once formula is not possible according to lemma 1. In this work we will extend the approach proposed by Lee et al [11]. The original approach by Lee is described in this section. The modification for treating incompletely specified functions is described in Section 5. Several approaches [9-11] have been proposed to identifying read-once functions (ROF). Golumbic described the current state-of-the-art algorithm. The IROF [9] algorithm needs an irredundant-sum-of-product (ISOP) as input and produces read-once formulae if it is possible or reports a failure otherwise. Despite the efficiency of the IROF algorithm, it cannot be modified to deal with incomplete-specified functions (ISF), since it depends on an ISOP as input. This is the reason why we choose to extend the approach by Lee et al [11].

The read-once algorithm proposed by Lee et al is based on some very simple observations given by the following Lemmas. For explaining these Lemmas, we consider all variables positive unate without loss of generality.

**Lemma 4:** if a read once formula exist for a function depending on more than one variable, at least one pair of variables can be tied together by a "+" or by a "*" operator.

**Lemma 5:** the tied variables described in lemma 4 can be substituted by a new single variable.

**Lemma 6:** When two variables *a* and *b* can be tied together by a "+" operator, the positive cofactors of both variables are equal.

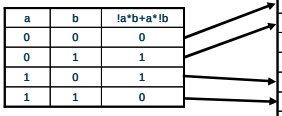
**Lemma 7:** When two variables *a* and *b* can be tied together by a "*" operator, the negative cofactors of both

variables are equal.

The approach proposed by Lee et al [11] ties two variables at a time substituting them for a single variable. The process is repeated until just one variable remains (in this case a read-once formula has been found) or the association is no longer possible (in which case the function is not read-once).

# 4. Domain transformation

According to Lemma 3, binate functions do not have read-once formulas. However, still according to lemma 3, a function is minimal if each polarity of a binate variable is read at most once. This leads us to the definition of read-polarity-once (RPO) functions. A RPO function read each polarity of a variable at most once in an optimized equation. An example of an RPO function is a two input exclusive or, which can be given by equations *!a*b+a*!b* and *(!a+!b)*(a+b)*. Notice that each variable is read at most once with each polarity. Strangely, the two equations are equal, but if they are rewritten as *na*b+a*nb* and *(na+nb)*(a+b)*, they become different logic functions. Notice that by introducing variables *na* and *nb*, a domain transformation is performed and the function become a 4-input function, with most of the lines appearing as don't cares. This is illustrated in Figure 1.



| a | b | !a*b+a*!b |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

| a | na | b | nb | !a*b+a*!b |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | X |
| 0 | 0 | 0 | 1 | X |
| 0 | 0 | 1 | 0 | X |
| 0 | 0 | 1 | 1 | X |
| 0 | 1 | 0 | 0 | X |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | X |
| 1 | 0 | 0 | 0 | X |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | X |
| 1 | 1 | 0 | 0 | X |
| 1 | 1 | 0 | 1 | X |
| 1 | 1 | 1 | 0 | X |
| 1 | 1 | 1 | 1 | X |

Fig. 1. Expanding a 2-input xor to 4 variables.

| a | na | b | nb | !a*b+a*!b | f(a=1) | f(a=0) | f(na=1) | f(na=0) | f(b=1) | f(b=0) | f(nb=1) | f(nb=0) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | X=0 | X | X | x | x | 0 | x | 0 | 0 |
| 0 | 0 | 0 | 1 | X=0 | 1 | X | 0 | x | x | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | X=0 | 0 | X | 1 | 0 | 0 | x | x | 0 |
| 0 | 0 | 1 | 1 | X | X | X | x | x | x | 0 | x | 0 |
| 0 | 1 | 0 | 0 | X=0 | X | X | x | x | 1 | x | 0 | x |
| 0 | 1 | 0 | 1 | 0 | X | 0 | 0 | x | x | 0 | 0 | x |
| 0 | 1 | 1 | 0 | 1 | X | 1 | 1 | 0 | 1 | x | x | 1 |
| 0 | 1 | 1 | 1 | X=1 | X | X | x | x | x | 0 | x | 1 |
| 1 | 0 | 0 | 0 | X=0 | X | X | x | x | 0 | x | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | X | x | 1 | x | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | X | 1 | 0 | 0 | x | 1 | 0 |
| 1 | 0 | 1 | 1 | X=1 | X | X | x | x | x | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | X | X | X | x | x | 1 | x | 1 | x |
| 1 | 1 | 0 | 1 | X=1 | X | 0 | x | 1 | x | 1 | 1 | x |
| 1 | 1 | 1 | 0 | X=1 | X | 1 | 1 | 0 | 1 | x | 1 | 1 |
| 1 | 1 | 1 | 1 | X=1 | X | X | x | x | x | 1 | 1 | 1 |

Fig. 2. Cofactors that are set to force the function to become positive unate.

The function represented by the truth table in Fig. 1 is not positive unate. By computing cofactors of the function and setting values to force the function to become positive unate in all of its variables, a new function is obtained. The computation of the co-factors and the new obtained function is shown in Figure 2. Notice that two unspecified lines remained unspecified after the process. This is directly related to the two different equations obtained for the xor2. Notice that the don't cares are set differently to obtain two possible different read once formulas for the 4-input function where presented initially.

| a | na | b | nb | na*b+a*nb |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

| a | na | b | nb | (na+nb)*(a+b) |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Fig. 3. How the remaining don't cares produce different read-polarity-once formulas.

The domain transformation proposed here is composed of two distinct steps. The first step expands a single variable (e.g *a*) into two separate variables (e.g. *a* and *na*). The first step is illustrated in Figure 1, for a 2-input exor. Notice that all the invalid combinations are set to don't care. Only the original valid values remain set to 0 or 1. The second step has the purpose to guarantee that all introduced variables have positive unate polarity. The second step is illustrated in Figure 2, for a 2-input exor. Cofactors are computed for every variable and don't cares are set in such a way that the polarity of the variable is guaranteed to be positive unate. This way, the two polarities of original binate variables are split into two separate variables with positive polarity. The resulting function is an incompletely specified function, for which one or more potential read-once formulae can be found. For this reason we introduce an algorithm to factor read once formula for incompletely specified functions in the next section.

# 5. Factoring ISRO functions

The read-once algorithm proposed by Lee et al [11] is based on some very simple observations given by the Lemmas explained in section 3. To expand the algorithm proposed by Lee et al to consider incompletely specified functions, we propose a new set of Lemmas. Again, we consider all variables positive unate without loss of generality; especially because the variables are forced to become positive unate by the second step of the domain transformation proposed in the previous section.

**Lemma 8:** if a read once formula exist for an incompletely specified function depending on more than one variable, at least one pair of variables can be tied together by a "+" or by a "*" operator.

**Lemma 9:** the tied variables described in lemma 8 can be substituted by a new single variable.

**Lemma 10:** When two variables *a* and *b* can be tied together by a "+" operator, the positive cofactors of both variables can be made equal by adequately setting don't cares.

**Lemma 11:** When two variables *a* and *b* can be tied together by a "*" operator, the negative cofactors of both variables can be made equal by adequately setting don't cares.

Our approach is a modified version of the approach proposed by Lee et al [11], that ties two variables at a time setting some don't cares and substituting them for a single variable. The process is repeated until just one variable remains (in this case a read-once formula has been found in the expanded set of variables) or the association is no longer possible (in which case the function is not read-once). Notice that finding a read once formula in the expanded set of variables, after the domain transformation, is equivalent to find a read-polarity-once formula for the variables before the domain transformation.

# 6. Complete algorithm

The complete algorithm proceeds in two steps. The first step reads a function and computes the polarity of the variables. Every binate variable is split into two separate positive unate variables according to the domain transformation presented in section 4. The second step performs the search for a read once formula for the incompletely specified function resulting from the domain transformations. The read once formula is then rewritten as a function of the original variables before the domain transformation.

# 7. Results

This section presents results to prove the efficiency of the proposed algorithm. In a first experiment, the set of all functions up to five inputs was studied. These functions were grouped into NPN (negation-permutation-negation) equivalence classes for enumeration feasibility. The total number of functions studied is 616125. To run the algorithm for all these functions 4 min of execution time were needed. The worst case optimization was 800ms and the average case was less than 1ms. Out of these 616125 functions, 1462 functions were read-polarity-once, while only 21 were read-once. Interestingly, the universe of read-once functions seems to be very limited compared to the universe of read-polarity-once functions. Our results demonstrate that the universe of RPO is quite broader than the universe of RO functions, for which many works have been devoted [9-12].

Comparative results for the quality of the algorithm are shown in Table 3, restricted to the set of 1462 RPO functions. The proposed algorithm performed better in terms of literals when compared to Quick factor [16], Good Factor [16], ABC [17] and XFactor [3,4]. The gain in terms of number of literals compared to X-factor is not very significant. However, the proposed algorithm guarantees exactness for Read-Polarity-Once functions.

Tab III - Number of literals for different approaches

| QF [16] | GF [16] | ABC [17] | X-FACTOR [3,4] | RPO (THIS PAPER) |
|---|---|---|---|---|
| 16086 | 15671 | 15981 | 13253 | 13064 |

In a second experiment, we have mapped all the benchmarks investigated by the Xfactor paper [4]. For the function cm163a_r Xfactor has found a 13 literal solution, while our algorithm found a 12 literal solution which is a RPO solution. This benchmark has seven inputs.

In a third experiment, we have investigated all circuits from ISCAS 85 [21]. The main idea of this experiment consists in evaluate how abundant are the RPO functions in the universe of industrial benchmarks. Since all RO functions are also RPO functions, we provide results for both classes. In order to do that, we use as input the mapped circuit (Verilog). A partition algorithm returns an AIG (And Inverter Graph) of this circuit, and a K-Cut [20] algorithm (with K=6) was performed in order to return the Boolean functions that appears in the mapped circuit

Tab IV – Abundance of functions in ISCAS 85 circuits.

| | K=2 | | K=3 | | K=4 | | K=5 | | K=6 | |
|---|---|---|---|---|---|---|---|---|---|---|
| CIRCUIT | R.O. | R.P.O | R.O. | R.P.O | R.O. | R.P.O | R.O. | R.P.O | R.O. | R.P.O |
| s1196 | 0,75 | 1,00 | 0,65 | 1,00 | 0,46 | 0,88 | 0,49 | 0,86 | 0,53 | 0,88 |
| s1238 | 0,75 | 1,00 | 0,55 | 1,00 | 0,49 | 0,89 | 0,49 | 0,85 | 0,52 | 0,85 |
| s13207 | 0,60 | 1,00 | 0,42 | 1,00 | 0,36 | 0,96 | 0,45 | 0,94 | 0,51 | 0,90 |
| s1423 | 0,60 | 1,00 | 0,43 | 0,96 | 0,35 | 0,92 | 0,40 | 0,89 | 0,44 | 0,87 |
| s1488 | 0,75 | 1,00 | 0,46 | 0,96 | 0,39 | 0,89 | 0,44 | 0,84 | 0,45 | 0,81 |
| s1494 | 0,75 | 1,00 | 0,46 | 0,96 | 0,38 | 0,88 | 0,41 | 0,82 | 0,44 | 0,79 |
| s15850 | 0,60 | 1,00 | 0,35 | 0,94 | 0,37 | 0,91 | 0,40 | 0,89 | 0,41 | 0,88 |
| s208 | 0,75 | 1,00 | 0,82 | 1,00 | 0,94 | 1,00 | 0,97 | 1,00 | 0,95 | 1,00 |
| s27 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 |
| s298 | 0,75 | 1,00 | 0,71 | 1,00 | 0,73 | 1,00 | 0,76 | 0,96 | 0,69 | 0,90 |
| s344 | 0,75 | 1,00 | 0,50 | 1,00 | 0,44 | 0,90 | 0,37 | 0,79 | 0,36 | 0,72 |
| s349 | 0,60 | 1,00 | 0,44 | 0,88 | 0,36 | 0,75 | 0,32 | 0,65 | 0,28 | 0,57 |
| s35932 | 0,60 | 1,00 | 0,37 | 0,89 | 0,29 | 0,84 | 0,24 | 0,72 | 0,22 | 0,70 |
| s382 | 0,60 | 1,00 | 0,65 | 1,00 | 0,61 | 0,95 | 0,62 | 0,95 | 0,59 | 0,92 |
| s38417 | 0,75 | 1,00 | 0,40 | 0,96 | 0,25 | 0,90 | 0,18 | 0,84 | 0,16 | 0,80 |
| s38584 | 0,71 | 1,00 | 0,42 | 0,95 | 0,24 | 0,92 | 0,20 | 0,87 | 0,21 | 0,84 |
| s386 | 1,00 | 1,00 | 0,85 | 1,00 | 0,80 | 0,96 | 0,73 | 0,88 | 0,67 | 0,81 |
| s400 | 0,60 | 1,00 | 0,65 | 1,00 | 0,63 | 0,96 | 0,67 | 0,97 | 0,61 | 0,97 |
| s420 | 0,60 | 1,00 | 0,70 | 1,00 | 0,80 | 1,00 | 0,75 | 1,00 | 0,69 | 1,00 |
| s444 | 0,75 | 1,00 | 0,58 | 1,00 | 0,71 | 1,00 | 0,66 | 0,97 | 0,62 | 0,93 |
| s510 | 0,75 | 1,00 | 0,71 | 1,00 | 0,67 | 0,97 | 0,62 | 0,93 | 0,50 | 0,85 |
| s526 | 0,60 | 1,00 | 0,57 | 1,00 | 0,67 | 0,95 | 0,59 | 0,87 | 0,60 | 0,86 |
| s526 | 0,60 | 1,00 | 0,50 | 1,00 | 0,60 | 1,00 | 0,57 | 0,94 | 0,59 | 0,93 |
| s5378 | 0,67 | 1,00 | 0,45 | 0,95 | 0,34 | 0,92 | 0,35 | 0,87 | 0,39 | 0,85 |
| s641 | 1,00 | 1,00 | 0,85 | 1,00 | 0,86 | 0,98 | 0,88 | 0,96 | 0,87 | 0,95 |
| s713 | 1,00 | 1,00 | 0,85 | 1,00 | 0,81 | 1,00 | 0,81 | 0,99 | 0,80 | 0,96 |
| s820 | 0,75 | 1,00 | 0,61 | 0,94 | 0,65 | 0,94 | 0,67 | 0,94 | 0,66 | 0,90 |
| s832 | 0,75 | 1,00 | 0,76 | 1,00 | 0,63 | 0,94 | 0,63 | 0,93 | 0,61 | 0,92 |
| s838 | 0,60 | 1,00 | 0,69 | 1,00 | 0,84 | 1,00 | 0,80 | 1,00 | 0,77 | 1,00 |
| s9234 | 0,71 | 1,00 | 0,47 | 0,94 | 0,38 | 0,90 | 0,39 | 0,87 | 0,37 | 0,83 |
| AVG | 0,72 | 1,00 | 0,60 | 0,98 | 0,57 | 0,94 | 0,56 | 0,90 | 0,55 | 0,87 |

RPO functions appears in average 94% of the total functions on the ISCAS circuits, while RO functions represents in average only 60% of the functions. It is possible to see in the table IV that there are circuits with 100% of RPO functions. Since this in an ongoing work, these cases will be better investigated on.

We have also investigated if mapped circuits could drive the results to a wrong way doing the synthesis of the AIG from BLIF file instead of the mapped circuit. This exercise shows the same behavior of our previously results. In Fig. 4 it is possible to see results for one specific circuit (s38584).

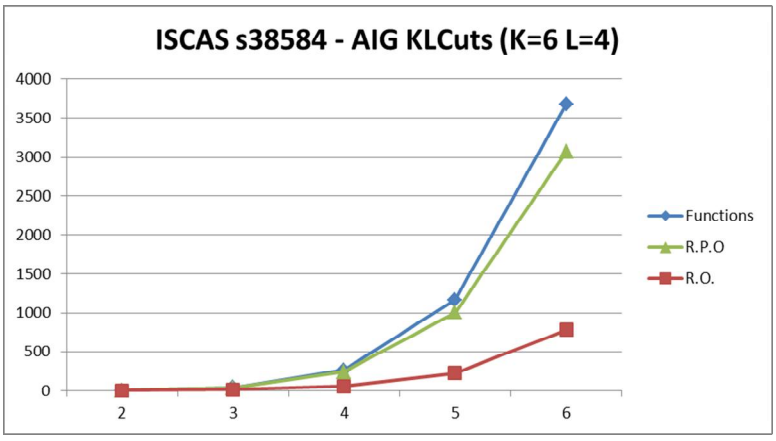



Fig. 4. Abundance of RO and RPO functions on the circuit s38584.

## 8. Conclusions

This paper has presented the concept of read-polarity once functions. Besides introducing the class of read-polarity-once functions, several related contributions were also introduced in this paper. Main contributions include: (1) an algorithm for factoring incompletely specified functions into Read-Once equations; (2) a domain transformation that splits existing binate variables into two independent unate variables; and (3) a complete algorithm for exact factoring of read-polarity-once functions. This algorithm was implemented and compared against state of the art factoring algorithms. The proposed algorithm presented better results in terms of quality and in terms of runtime.

Out of 616125 5-input studied functions, 1462 functions were read-polarity-once, while only 21 were read-once. Interestingly, the universe of read-once functions seems to be very limited compared to the universe of read-polarity-once functions. Our investigation under benchmark ISCAS demonstrate that the universe of RPO is quite broader than the universe of RO functions, for which many works have been devoted [9-12].

## 9. Acknowledgements

Research partially funded by Nangate Inc. under a Nangate/UFRGS research agreement, by CNPq Brazilian funding agency, by FAPERGS under grant 11/2053-9 (Pronem)., and by the European Community's Seventh Framework Programme under grant 248538 – Synaptic.

## 10. References

[1] R. K. Brayton. 1987. Factoring logic functions. *IBM J. Res*. Dev. 31, 2 (March 1987), 187-198.

[2] Hachtel, G. D. and Somenzi, F. 2000. *Logic Synthesis and Verification Algorithms* (1st ed.). Kluwer Academic Publishers, Norwell, MA, USA.

[3] Golumbic, M. C. and Mintz, A. 1999. Factoring logic functions using graph partitioning. In *Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design (ICCAD '99)*. IEEE Press, Piscataway, NJ, USA, 195-199.

[4] Mintz, A. and Golumbic, M. C. 2005. Factoring Boolean functions using graph partitioning. *Discrete Applied Mathematics*, Volume 149, Issues 1–3, Pages 131-153.

[5] Stanion, T. and Sechen, C. 1994. Boolean division and factorization using binary decision diagrams. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol.13, no.9, pp.1179-1184, Sep 1994.

[6] Lawler, E. L. 1964. An Approach to Multilevel Boolean Minimization. *J. ACM* 11, 3 (July 1964), 283-295.

[7] Yoshida, H.; Ikeda, M. and Asada, K. 2006. Exact Minimum Logic Factoring via Quantified Boolean Satisfiability. *Electronics, Circuits and Systems, 2006. ICECS '06. 13th IEEE International Conference on*, vol., no., pp.1065-1068, 10-13 Dec. 2006.

[8] Yoshida, H. and Fujita M. 2011. Exact Minimum Factoring of Incompletely Specified Logic Functions via Quantified Boolean Satisfiability. *IPSJ Transactions on System LSI Design Methodology*, Vol. 4, pp. 70-79, Feb. 2011.

[9] Golumbic, M. C.; Mintz, A. and Rotics, U. 2001. Factoring and recognition of read-once functions using cographs and normality. In *Proceedings of the 38th annual Design Automation Conference (DAC '01)*. ACM, New York, NY, USA, 109-114.

[10] Golumbic, M. C.; Mintz, A. and Rotics, U. 2008. An improvement on the complexity of factoring read-once Boolean functions. *Discrete Appl. Math*. 156, 10 (May 2008), 1633-1636.

[11] Lee, T. and Wang, C. 2007. Recognition of Fanout-free Functions. *Design Automation Conference, ASP-DAC '07. Asia and South Pacific*, vol., no., pp.426-431, 23-26 Jan. 2007.

[12] Golumbic, M. C. and Gurvich, V. A. *Read-once functions*. Crama, Y and Hammer, P. L. 2008. *Boolean Functions: Theory, Algorithms and Applications*. Cambridge University Press, Cambridge, MA (2008) (Chapter 12).

[13] Mishchenko, A.; Chatterjee, S. and Brayton, R. 2006. DAG-aware AIG rewriting a fresh look at combinational logic synthesis. In *Proceedings of the 43rd annual Design Automation Conference (DAC '06)*. ACM, New York, NY, USA, 532-535.

[14] H.S. Warren. *Hacker's Delight*. Addison-Wesley Professional, 2002.

[15] Elbassioni, K.; Makino, K. and Rauf, I. 2009. On the readability of monotone Boolean formulae. *Journal of Combinatorial Optimization*.Volume 22, Number 3, 293-304.

[16] Sentovich, E.; Singh, K.; Lavagno, L.; Moon, C.; Murgai, R.; Saldanha, A.; Savoj, H.; Stephan, P.; Brayton, R. and Sangiovanni-Vincentelli, A. L. 1992. SIS: A system for sequential circuit synthesis. *Tech. Rep. UCB/ERL M92/41*. UC Berkeley, Berkeley. 1992.

[17] Berkeley Logic Synthesis and Verification Group, ABC: A System for Sequential Synthesis and Verification, Release 051205. http://www.eecs.berkeley.edu/~alanmi/abc/

[18] Martins, M.G.A.; da Rosa Junior, L. S.; Rasmussen, A.B.; Ribas, R.P. and Reis, A.I. 2010. Boolean factoring with multi-objective goals. *Computer Design (ICCD), 2010 IEEE International Conference on*, vol., no., pp.229-234, 3-6 Oct. 2010.

[19] Werber, J.; Rautenbach, D. and Szegedy, C. 2007.Timing optimization by restructuring long combinatorial paths. *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, vol., no., pp.536-543, 4-8 Nov. 2007.

[20] Martinello, O. Jr.; Marques, F. S.; Ribas, R. P. and Reis, A. I. 2010. KL-cuts: a new approach for logic synthesis targeting multiple output blocks. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '10)*. European Design and Automation Association, 3001 Leuven, Belgium, Belgium, pp.777-782.

[21] IWLS 2005 Benchmarks. http://www.iwls.org