J Korea Industr Inf Syst Res Volume 18 Number 1
http://dx.doi.org/10.9723/jksiis.2013.18.1.025

Derivations of Single Hypothetical Don't-Care Minterms Using the Quasi Quine-McCluskey Method

Eungi Kim*

Abstract Automatically deriving only individual don't-care minterms that can effectively reduce a Boolean logic expressions are being investigated. Don't-care conditions play an important role in optimizing logic design. The type of unknown don't-care minterms that can always reduce the number of product terms in Boolean expression are referred as single hypothetical don't-care (S-HDC) minterms. This paper describes the Quasi Quine-McCluskey method that systematically derives S-HDC minterms. For the most part, this method is similar to the original Quine-McCluskey method in deriving the prime implicants. However, the Quasi Quine-McCluskey method further derives S-HDC minterms by applying so-called a combinatorial comparison operation. Upon completion of the procedure, the designer can review generated S-HDC minterms to test its appropriateness for a particular application.

Key Words : minterm, don't-care condition, Quine-McCluskey method, Karnaugh Map

1. Introduction

The number of product terms in Boolean expressions is closely related to logic-based designs [1]. Because of this, the designers often face the challenge of finding the most efficient way to optimize and manipulate logic based designs. In doing so, the process can involve not only finding an equivalent logic expression, but it can also involve identifying specific conditions in which Boolean expressions can be even further simplified.

These types of elements in a logical design could be considered as a "degree of freedom" [2][3][8]. In such cases, a user is allowed to optimize the specified design based on the degree of freedom. Therefore, exploring alternative solutions is desirable since it may produce optimal Boolean expressions at the end.

More specifically, such a degree of freedom is usually specified in terms of "don't-care conditions". In other words, don't-care conditions are basically certain cases where input or output conditions can never occur.

Until recently, different types of don't-care conditions have been identified in literature [4][5][6][7][8]. In almost all instances, don't-cares are usually provided by the user or derived from a particular environment. They are also derived or observed from particular interconnection of circuit logic gates.

These conditions should be properly identified and labeled as a don't-care condition during the initial design stage. Provided that a Boolean expression is in sum of product (SOP) form, the proposed method finds special type of don't-care minterms.

* Department of Information Communication Namseoul University, South Korea(e-mail: eungikim68@daum.net)

The type of don't-care minterms which this method is concerned with is one that can always combine with some other Boolean expression product terms. In effect, it produces less product terms in the end. This special type of don't-care conditions is referred as a single hypothetical don't-care (S-HDC) minterm in this paper. Unlike a typical don't-care minterm, each S-HDC minterm is guaranteed to minimize existing Boolean logic expressions by eliminating at least one product term.

In pursuit of deriving S-HDC minterms, we examine the applicability of the Karnaugh Map [9] and the Quine-McCluskey method [10]. These Boolean minimization methods, which are extremely popular, are relatively simple to use. These techniques are widely discussed in digital design books such as [1][11][12]. Until recently, there have been various attempts to explore both methods in order to extend and optimize these techniques [13][14][15]. In this paper, the original Quine-McCluskey method [10] was slightly modified for ease of use. At the same time, the method has been modified in order to derive S-HDC minterms. The advantage of the Quine-McCluskey method is that it is systematic, and it can handle a greater number of variables compared to the Karnaugh Map.

The proposed method that derives S-HDC minterms consists of two separate parts. The first part comprises of a method to minimize Boolean expressions. The method is basically based on the Quine-McCluskey method. The second part comprises of a method to derive S-HDC minterms based first part's result. Collectively, the author calls the procedures as the Quasi Quine-McCluskey method, and the method is described subsequently.

2. Definitions

The following terms are essential in order to understand the author's Quasi Quine-McCluskey method. These terms will be used throughout this paper.

Implicant minterm is a minterm which belongs to a finite group of literals. Each implicant is a product of literals. For example, if x_n and x_{n+1} are implicant minterms, then the difference between x_n and x_{n+1} must be a power of 2.

Implicant group is a groups of minterms which are used to calculate prime implicants and derive S-HDC minterm derivation. It is expressed as (x_n, x_{n+1}) d_i.

M-diff (minterm difference) is the differing values between member minterms of an implicant group. For an implicant with two member minterms, M-diff, denoted as d_i is the difference between x_1 and x_2.

Member minterms refer to minterms which belong to an implicant group. The size of member minterm expands by a factor of 2 in accordance with iteration increase.

Prime implicant is a type of implicant that no longer remains as an implicant if any of its literals in product terms is removed.

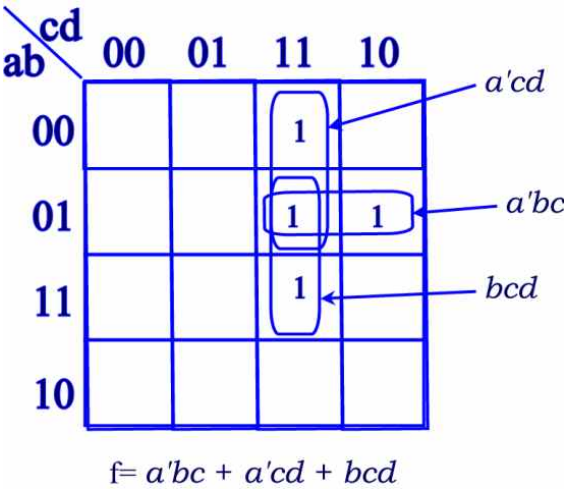
Combinatorial Comparison Operation is an operation that is performed to derive candidate S-HDC minterms. Combinatorial comparison operation refers to the combinatorial based operations of addition and subtraction. The operation specifically involves unmatched member minterms of implicant group with its M-diff values.

3. The Karnaugh Map Method

First, a detailed understanding of the notion of S-HDC minterm is useful prior to understanding the S-HDC derivation method. For this purpose, the Karnaugh Map method is useful for discussing the idea behind the S-HDC minterms. The Karnaugh Map is a widely used Boolean

minimization method. Because of its simplicity of representation, we can use the Karnaugh Maps to identify S-HDC minterms graphically. For an illustration, consider a Boolean function problem Σ (3,6,7,15). The Karnaugh Map representation of these minterms and the minimized Boolean expression are shown in Figure 1.

Since S-HDC minterms have the special characteristics of combining to the existing minterms, there are cells on this map which would have combined if they were hypothetically marked as "1". S-HDC minterms are marked with "X" on the Karnaugh Map with an enclosing circle. As shown in Figure 2, for this particular Boolean function, three S-HDC minterms are possible .

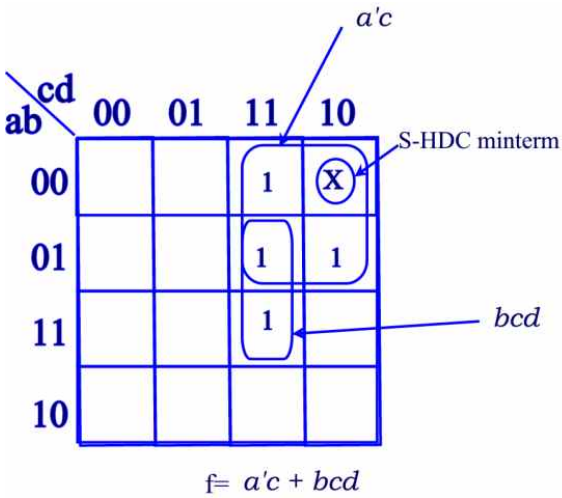


<Figure 1> Karnaugh Map Representation of Σ(3,6,7,15)

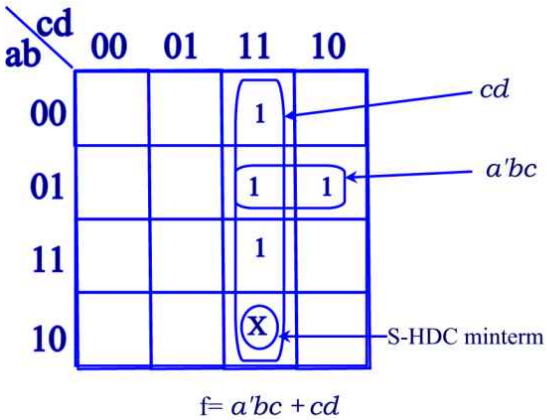
In each case, if S-HDC minterms are used, the final Boolean expression will end-up with fewer literals. The reason is that as the group cell size of Karnaugh Map increases, the number of literals in each product term will decrease proportionally.

For example, for a 4-variable Boolean function, if groups of 4 member cells form a prime implicant, then the number of literals in a term is 3. If a group of 8 member cells form a prime implicant, then the number of literals in a product term is 2. Because a S-HDC minterm plays the role of combining other implicant groups, it reduces the number of product terms.

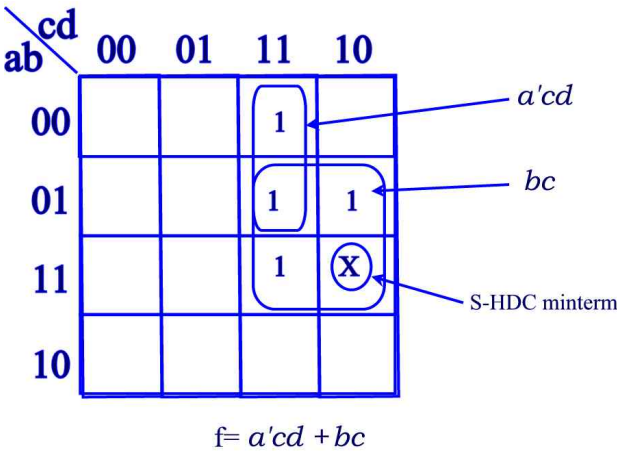
(a) S-HDC Minterm Value 12



(b) S-HDC Minterm Value 11



(c) S-HDC Minterm Value 14



<Figure 2> Karnaugh Map with S-HDC minterms

In essence, for *n* number of Boolean variable problems, 2^n combinations of true conditions can be marked as "1" on the Karnaugh Map. Based on this map, remaining cells can potentially become a S-HDC minterm. Using the Karnaugh Map, it is a relatively trivial task to visually spot S-HDC minterms. The limitation of the above Karnaugh Map approach is that it is limited to Boolean functions that have small number of variables. For larger variable problems, a more systematic approach is necessary in order to find S-HDC minterms.

4. The Quasi Quine-McCluskey Method Part I

The first part of this method is to minimize a given Boolean function problem. The first part of the method closely resembles the Quine-McCluskey method as shown in [10]. Since the original Quine-McCluskey method is less efficient in terms of comparing implicants, an improved version of the Quine-McCluskey method is suggested here. The first part of Quasi Quine-McCluskey Method method is similar to the method suggested by [14]. Let us consider the Boolean function ∑(3,6,7,15) once again. The following procedures are essential steps for the first part.

Step 1. *Place minterms into different blocks based on the number of binary 1's.* As in the Quine-McCluskey method, minterms based on number of 1's in the binary form need to be grouped into different blocks. In this paper, we refer to these types of groups as a *bit-numbered block*. For example, in the left side of Table 1, minterms for the Boolean function ∑(3,6,7,15) are grouped together as individual bit-numbered blocks.

Step 2. *Starting from the lowest bit numbered block, group the minterms by comparing each minterm in one block to adjacent bit numbered blocks.* A valid implicant minterm is formed only if the M-diff is a power of 2. M-diff is the differing values between member minterms of an implicant group. Thus, a valid implicant group in Iteration I would consist of two member minterms, and their differing value would be 1, 2, 4, etc. The result of iteration is shown on the right side of Table 1.

For example, as a result of the first comparison, the implicant minterm that is produced is (3,7)4. In this case, 4, which is the value outside of the parenthesis, is the M-diff. Note the M-diff is valid because the difference is a power of 2. In terms of a binary number, a valid M-diff indicates that two member minterms differ by only one bit.

The result shown in this table is produced by comparing every minterms in each bit-numbered block to its adjacent block. When comparing, the minterm values in the next higher bit-numbered block needs to be higher than the minterm values in the current bit-numbered block. If it is not higher, then an implicant group cannot be formed, and a M-diff value does not have to be obtained since the procedure can be skipped.

<Table 1> Iteration I Result for Σ(3,6,7,15)

Bit-Numbered Block #	Min-term
1	3 6
2	7
3	15

Bit-Numbered Block #	Iteration I
1	(3,7) 4 ** (6,7) 1 **
2	(7,15) 8 **

Step 3. *Continue to compare minterm differences from each block to identify prime implicants until all groupings become complete.* As in the Quine-McCluskey method, the iterations need to continue until all prime implicants can are derived. In this example, since each minterm from one block is different from its adjacent block, no further iteration is required. This procedure derived three prime implicants as a result. Each prime implicants are marked with "**" in Table 1.

Step 4. *Use a prime implicant chart to select prime implicants.* The Quine-McCluskey method requires a prime implicant chart in order to find minimal amount of covering prime implicants. The Quasi Quine-McCluskey method is not any different in this respect. The implicant chart for this problem is shown in Table 2. The goal in this step is to find a minimum amount of prime implicants to cover every minterm. In this case, all of the prime implicants must be selected to cover the minterms which are listed on the top of the chart. Hence, all of the prime implicants need to be selected in order to cover the minterms which are placed on the top horizontal cells in the table.

Step 5. *Translate the selected prime implicants to a Boolean final expression.* Once prime implicants are selected, the final result which is in implicant minterm expression form should be rewritten in a Boolean expression form. The translation can be done by intersecting the largest member value from the selected prime implicant with its M-diff values.

<Table 2> Prime Implicant Chart for Σ(3,6,7,15)

	3	6	7	15
(3,7) 4	X		X	
(6,7) 1		X	X	
(7,15) 8			X	X

For example, the largest member minterm for the prime implicant (3,7)4 is 7, and the M-diff value is 4. Thus, "0111" intersecting "0100" would result in "0x11". In essence, the "x" mark is used to cross out the intersected 1's. "0x11" is equivalent to Boolean product term *a'cd*. By converting all of the remaining binary numbers back to Boolean product terms, we end up with *a'cd+a'bc+bcd* in the end.

The proposed method differs from the Quine-McCluskey method in a number of ways. First, a decimal based scheme rather than a binary based scheme is being used. Second, the proposed method also avoids bit unnecessary in Iteration I. The reason is that a comparison is necessary only if minterms in higher bit-numbered block is greater than the minterms in the lower adjacent bit-numbered block. Lastly, in the higher iterations, only M-diff values need to be matched instead of comparing every implicant.

5. The Quasi Quine-McCluskey Method Part II

The second part of the Quasi Quine-McCluskey Method is to perform combinatorial comparison operations in order to derive S-HDC minterms. In this part, combinatorial comparison operations are sequentially performed to generate all possible S-HDC minterms. This list can be produced by comparing prime implicants within each block and also by comparing prime implicants to adjacent block.

For Boolean function Σ(3,6,7,15), only one iteration is required. In Iteration I, prime implicants

are (3,7)4, (6,7)1 and (7,15)8. Formally, two implcant minterms (x_1,x_2)d_1 and (y_1,y_2)d_2 can be combined to form by following combinatorial comparison operations: if one of these conditions (x_1=y_1 or x_1=y_2 or x_2=y_1 or x_2=y_2) is true then, for each unmatched member minterm, generate results by adding-subtracting the M-diff from opposing side of implicant group and the add M-diff from opposing implicant group side. If two results are equal, then the result is the S-HDC mitnerm.

For a detailed illustration of this procedure, consider the pseudo code shown in Figure 3. First, let us use variables *a* and *b* as temporary array values. Two minterm members in an implicant group imply that we need four arrays of *a* and *b*. If we consider only a case where two member minterms x_1 and y_1 having equal values, then the possibilities can be specified using an algorithm shown in this figure.

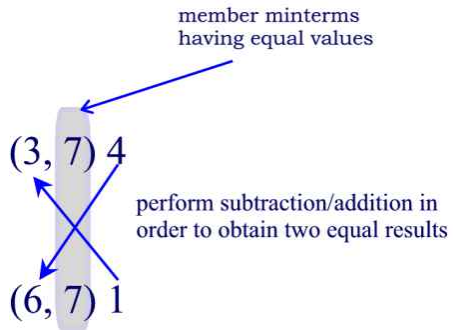
```
if x1=y1

{
        a[1]=x2 - d2; b[1]= x2 - d1;
        a[2]=x2 + d2; b[2]= x2 - d1;
        a[3]=y2 - d2; b[3]= y2 + d1;
        a[4]=y2 + d2; b[4]= y2 + d1;
}
function_mm (a,b);   /* function call */
..
..
func_mm (int a[4], int b[4]);

for (int i=0; i < 4; i++)
{
        for (int j=0; j < 4; j++)
        {
                if (a[i]=b[j])
                cand_mm=i;
        }
}
```

<Figure 3> A Pseudo Code for x_1=y_1 Condition

This pseudo code is applicable for the condition where x_1=y_1. Additionally, similar steps need to be taken for the condition x_1=y_2, x_2=y_1 and x_2=y_2. In each case, the algorithm would need to call the function *func_mm* in order to derive a correct S-HDC minterm. Also notice that this code is applicable only for Iteration I.

For a graphical illustration of this procedure, consider Figure 4. This figure represents two pairs of prime implicants which are being compared in order to derive S-HDC minterms. In a nutshell, if the results from one addition-subtraction operation is exactly the same as the result of another addition-subtraction operation, then the resulting number becomes a S-HDC minterm.

After attempting these combinatorial comparisons, if the resulting number is equal based on discrete operations, then the number which is equal is a S-HDC minterm. In this example, one of two results, (3-1) or (3+1), must be equal to another one of two results, (6-4) or (6+4). If (3-1) and (6-4) are selected, then the result of these operations are equal to each other since both operations produces "2". Thus, this "2" must be selected.



<Figure 4> Comparison Illustration of Two Pairs of Prime Implicants

A more detailed example of this procedure is also shown in Table 3. This table shows entire prime implicant minterms and their corresponding combinatorial comparison operations with other implicant minterms. As shown, the member minterm which are equal to another implicant minterm are underlined. In each implicant comparison, we need

to verify whether the member minterm are in any of implicant groups in the higher iteration. Since neither of these cases are applicable, only Iteration I is required.

<Table 3> The Combinatorial Comparison Operations and the S-HDC Minterm Calculation

Index #	Implicant Groups	S-HDC Minterm Calculation
1	(3,7)4 (6,7)1	3-1=6-4 2 is a S-HDC minterm
2	(3,7)4 (7,15)8	3+8=15-4 11 is a S-HDC minterm
3	(6,7)1 (7,15)8	6+8=15-1 14 is a S-HDC minterm
S-HDC Minterms		2,11,14

6. S-HDC Minterm Calculation: Skip or Perform

The previous example demonstrated a case where only one iteration was required in order to find all of prime implicants. Most often, however, additional iterations are required. The Boolean function Σ (0,1,3,4,5,7,12,13,15) requires two iterations as shown in Table 4. For Iteration II, all possible combinations without removing any duplicates should be generated first. Then, some of the implicant groups having only differences in M-diff values need to be combined. In Iteration II, unlike Iteration I, if the generated combination is not combinable, then it is not a valid implicant group. Therefore, it should be removed from the list.

In Table 4, the implicant groups (0,1,4,5)1 and (0,4,1,5)4 are combined, forming (0,1,4,5)1,4. Thus, (0,1,4,5)1 and (0,4,1,5)4 are both crossed out, indicating that they have been combined. Also, note that each prime implicant contains 4 member minterms instead of two minterms. Similarly, the implicant groups (1,3,5,7)2,4, (4,5,12,13)1,8, (5,7,13,15) 2,8 have been formed as a result of combining in Iteration II. Same as the previous example, each prime implicants is marked with "**" in this table.

<Table 4> Prime Implicants After Iteration I & II for Σ(0,1,3,4,5,7,12,13,15)

Minterm	Iteration I	Iteration II
0 1 4	(0,1)1 (0,4)4	~~(0,1,4,5)1~~ ~~(0,4,1,5)4~~ **(0,1,4,5)1,4 ****
3 5 12	(1,3)2 (1,5)4 (4,5)1 (4,12)8	~~(1,3,5,7)2~~ ~~(1,5,3,7)4~~ **(1,3,5,7)2,4 ****
7 13	(3,7),4 (5,7)2 (5,13)8 (12,13)1	~~(4,12,5,13)1~~ ~~(4,12,5,13)8~~ **(4,5,12,13)1,8 ****
15	(7,15)8 (13,15)2	~~(5,7,13,15)2~~ ~~(5,13,7,15)8~~ **(5,7,13,15)2,8 ****

After using the prime implicant chart as shown in Table 5, the simplified expression for this Boolean function is *a′c′+a′d+bc′+bd.*

<Table 5> Prime Implicants Chart

	0	1	3	4	5	7	12	13	15
(0,1,4,5)1,4	X	X		X	X				
(1,3,5,7)2,4		X	X		X	X			
(4,5,12,13)1,8				X	X		X	X	
(5,7,13,15)2,8					X	X		X	X

For the second part of the method, the process is slightly more complex for this Boolean function. The complete operations needed to derive S-HDC minterms are shown in Table 6.

Here, the combinatorial operations can be categorized into two cases:

a) *skip* the S-HDC calculation, and
b) *perform* the S-HDC calculation.

First, the skipping rule is as follows: In Interation II, if the implicant members are members of higher iteration implicant group, then S-HDC cannot be derived. The calculation procedure needs to be skipped, and we need to proceed to a next pair of implicant groups.

As shown in this example, (0,1)1 and (0,4)4 needs to be compared for the combinatoral comparion operations. Minterm "0" is the matching minterm since minterm "0" exist in both implicant groups. Note that the combinatorial comparion operation involves three individual minterms, and they need to be examined collectively. For the first operation, the three numbers are "0", "1" and "4". If an implicant group in the next higher iteration contains all of these minterms (0, 1, and 4), then comparison of the minterms and the S-HDC calculation operation can be skipped.

The second rule specifies a valid condition where the S-HDC calculation can be performed. Succinctly stated, if the implicant minterms are not member of a single implicant group in next higher iteration, then we perform additional S-HDC comparison procedures.

The additionally required procedure is the following: If two pairs are involved, we can take each pair of minterms and locate the matching implicant group in Iteration 2. Then, the unmatched pairs of minterms must be examined to see if they are member minterms of higher iteration implicant groups. If both pairs of unmatched minterms are members of a higher iteration implicant group, then the condition is valid for calculating a S-HDC minterm. If it is a valid condition, then we can perform a S-HDC minterm calculation based on combinatorial minterm operations.

<Table 6> The Derivation of Candidate S-HDC for Σ(0,1,3,4,5,7,12,13,15)

Index #	Implicant Pairs	S-HDC Minterm Calculation
1	(0,1)1 (0,4)4	Skip
2	(0,1)1 (1,3)2	Perform 2
3	(0,1)1 (1,5)4	Skip
4	(0,4)4 (4,5)1	Skip
5	(0,4)4 (4,12)8	Perform 8
6	(1,3)2 (3,7)4	Skip
7	(1,5)4 (5,7)2	Skip
8	(1,5)4 (5,13)8	Skip
9	(4,5)1 (5,7)2	Skip
10	(4,12)8 (12,13)1	Skip
11	(3,7)4 (5,7)2	Skip
12	(5,7)2 (5,13)8	Skip
13	(5,13)8 (12,13)1	Skip
14	(3,7)4 (7,15)8	Perform 11
15	(5,7)2 (7,15)8	Skip
13	(5,13)8 (13,15)2	Skip
14	(12,13)1 (13,15)2	Perform 14
15	(7,15)8 (13,15)2	Skip
S-HDC Minterms		2,8,11,14

For example, in Table 6, the first comparison implicant pairs are (0,1)1 and (0,4) 4. However, "0",

"1", and "4" exists in the implicant group (0,1,4,5). Since all of these hypothetical minterms are found in the higher implicant group, the S-HDC calculation can be skipped for this particular set of minterms. Now, for index #2, (0,1)1 and (1,3)2 are being compared. Notice that (0,1) is member minterms of implicant group (0,1,4,5). At the same time, the remaining implicant group (4,5) exists in the implicant group (4,5,12,13). In addition, (1,3) is a member minterms of the implicant (1,3,5,7) and the remaining minterms (5,7) is in implicant group (5,7,13,15). The resulting calculation can be performed since this is a valid case. After both addition and subtraction attempts, we see that a successful equal value can be obtained since 0+2=2 and 3-1=2. Since the result of these operations are equal, the S-HDC minterm for index #2 is "2". The result of the entire operations is shown on this table.

7. The Effects of S-HDC Minterm

After obtaining every S-HDC minterm, we can simply run the Quasi Quine-McCluskey method again to see the effects using the S-HDC minterm. Since "2", "8", "11", and "14" are S-HDC minterms, any of these minterm values can be added to the original Boolean function to see the effects. For instance, after adding S-HDC minterm "2", the new Boolean function would be $\sum(0,1,2,3,4,5,7,12,13,15)$. After running the first part of the Quasi Quine-McCluskey method, a minimized expression for this problem with the S-HDC minterm "2" is *a'b'+bc'+bd.* After adding S-HDC minterm "2" to the Boolean function, the minimized expression result is more concise than the minimized expression from the previous Boolean function $\sum(0,1,3,4,5,7,12,13,15)$. In effect, we are comparing the minimized result without any S-HDC minterm with the minimized results with a S-HDC minterm. Here, a new minimized Boolean expression resulted in 3 product terms instead of 4 product terms. The remaining S-HDC minterms "8", "11", and "14" will produce similar effects if they are individually added to the original Boolean function.

The illustration so far was limited to comparing 4 member minterms. However, the combinatorial comparison operation is basically the same for even higher iterations. Forming a group of 4 adjacent minterms is a prerequisite for forming a group 8 adjacent minterms. In other words, an implicant group of 8 requires two adjacent implicant groups that have 4 member minterms. If there are 7 adjacent minterms, then it can be viewed as 3 adjacent minterms, and an implicant group of 4. Since the method describe in this paper derives a S-HDC minterm based on recognizing 3 adjacent minterms, it has a minimizing effect on a group of 7 adjacent minterms as well. Therefore, the skip and valid procedures which was described earlier works for cases beyond Iteration II.

8. Conclusion and Future Works

By extending the Quine-McCluskey method, the proposed Quasi Quine-McCluskey method systematically identified every possible instances of S-HDC minterms. This paper demonstrated that the Quasi Quine-McCluskey method is effective in deriving a combinational set of minterms that can group together with existing minterms. The computer based implementation should be straight forward since the method described in this paper is systematic.

The suitability of S-HDC minterms are left up to a designer. Since they are purely hypothetical information from designer's standpoint, the designer has to make a decision whether these can be applied for a particular application. It should be stressed that the basic premise of using S-HDC minterms is that in certain situations changing or

sacrificing logical condition is acceptable in order to obtain a minimized Boolean expression.

For future works, a wider range of S-HDC minterm patterns involving larger variable problems can be considered. For example, instead of only considering "single" don't-care conditions, "dual" don't-care conditions can be considered as well. In such a case, a pair of don't-care minterms or a combination of both pair and S-HDC minterms would produce greater minimizing effects in general. Also, we could also consider cases, where minimization can be achieved by eliminating existing Boolean function minterms instead of deriving a S-HDC minterm. Obviously, the suggested method only solves a small part of imaginable don't-care minterms since there are many possible variations regarding hypothetical don't care minterms. Therefore, different types of hypothetical don't-care minterms and the effects of their patterns should be more throughly investigated in the future.

References

[1] C. H. Roth, Jr., *Fundamentals of Logic Design*, 6th ed., Thomson Engineering, 2009.

[2] D. Brand, R.A. Bergamaschi, and L. Stok, "Don't-cares in synthesis: Theoretical pitfalls and practical solutions," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 4, pp. 285-304, April, 1998.

[3] S. C. Chang and M. Marek-Sadowska, "An efficient algorithm for local don't-care sets calculation," in *Proc. of Design Automation Conf.*, 1995, pp. 663-667.

[4] H. Chou, K. Chang, and S. Kuo, "Accurately handle don't-care conditions in high-level designs and application for reducing initialized registers," *IEEE Transactions on CAD of Integrated Circuits and Systems* vol. 29, no. 4, pp. 646-651, 2010.

[5] A. Mishchenko, R. Brayton, J. Jiang, and S. Jang. "Scalable don't-care-based logic optimization and resynthesis," *ACM Trans. Reconfigurable Tech. Syst.* vol 4., no. 34, Dec. 2011.

[6] S. Safarpour, A.G. Veneris, and R. Drechsler, "Integrating observability don't-cares in all-solution SAT solvers," in *Proc. ISCAS*, 2006.

[7] M. Nosrati, R. Karimi, and R. Aziztabar, "Minimization of boolean functions which include don't-care statements, using graph data structure," in *Proc. of WiMoA 2011/ICCSEA* Dubai, UAE, 2012, pp. 212-220.

[8] M. Damiani and G. De Micheli, "Observability don't-care sets and Boolean relations," in *Proc. Int. Conf. Comput.-Aided Des.*, 1990, pp. 502-505.

[9] M. Karnaugh, "A map method for synthesis of combinational logic circuits," *Transactions of the AIEE, Communications and Electronics*, vol. 72, pp. 593-599, 1953.

[10] E. J. McCluskey, "Minimization of Boolean functions," *Bell System Tech. Journal*, vol. 35, no. 5, pp. 1417-1444, 1956.

[11] P. K. Lala, *Principles of Modern Digital Design*, John Wiley & Sons, Inc., Hoboken, NJ, USA, 2006.

[12] R. Katz and G. Borriello, *Contemporary Logic Design*, 2nd Ed., Pearson/Prentice Hall. 2005.

[13] P. W. C. Prasad, A. Beg, and A. K. Singh, "Effect of Quine-McCluskey simplification on Boolean space complexity," *IEEE Proceeding 2009 Conference on Innovative Technologies in Intelligent Systems and Industrial Applications*, pp. 165-170, Monash University, July, 2009.

[14] R. Mohan Ranga Rao, "An Innovative procedure to minimize Boolean function," *International Journal of Advanced Engineering Sciences and Technologies(IJAEST)*, vol. 3, no. 1, pp. 12-14, 2011.

[15] M. Petrík. "Quine-McCluskey method for many-valued logical functions," *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, vol. 12, no. 4, pp. 393-402, April, 2007.

김 은 기 (Eungi Kim)

- 정회원
- Indiana Univ. of Pennsylvania (BS) (Computer Science)
- Illinois State University (MS) (Applied Computer Science)
- University of North Texas (Ph.D Candidate in Information Science)
- 남서울 대학교 정보통신학과 외국인교수
- 관심분야: Information Retrieval, Digital Systems

논 문 접 수 일 : 2012년 11월 13일
1차수정완료일 : 2013년 01월 02일
2차수정완료일 : 2013년 01월 24일
게 재 확 정 일 : 2013년 01월 25일