ACSIJ Advances in Computer Science: an International Journal, Vol. 3, Issue 2, No.8 , March 2014
ISSN : 2322-5157
www.ACSIJ.org

ACSIJ
WWW.ACSIJ.ORG

Heuristics for Irreversible Ternary MVL Circuit Synthesis and Optimization Using PSO Algorithm

Maryam Yarmohammadi[1], Majid Mohammadi[2]

[1]**Department of Computer, Science and Research Branch, Islamic Azad University, Tehran, Iran**
Yarmohamadi_maryam@yahoo.com

[2]**Department of Computer Engineering, Shahid Bahonar Kerman University, Kerman, Iran**
mohammadi@mail.uk.ac.ir

Abstract

This article proposed some heuristics in synthesis and optimization of ternary irreversible logic circuit with Particle Swarm Optimization (PSO) algorithm. Some of benchmark irreversible circuits such as full adder and multiplier are synthesis using the method proposed in this research. Number of gate are merit to compare the performance of different irreversible logic circuit. The result, presented in this dissertation, indicate efficiency of the proposed method to synthesis of irreversible ternary circuit.

Keywords: *multiple valued irreversible circuits, optimization, PSO algorithm, irreversible circuit synthesis.*

1. Introduction

MVL synthesis issue is more troublesome contrasted with its binary partner. Consider, for example, synthesis of 3-variable 3-valued functions. There are 3^{27} such functions. Various heuristic algorithms for producing near-minimal sum-of products realization of MVL circuits have been introduced. Iterative heuristics offer the likelihood of investigating bigger result space in touching base at near-optimal results. Various these techniques have been accounted for in the written works [1]-[13].

In [7], application of PSO algorithm for synthesis of MVL functions is introduced and shown the better result of using PSO against other algorithms. In this paper, we extend the use of PSO algorithm for synthesis of MVL functions and specified it for ternary MVL with 3-inputs. The paper is organized as follows. In Section 2, we provide some background material for synthesis of MVL functions. In Section 3, a brief introduction to particle swarm optimization that introduced in [7] is provided. The proposed heuristics for the MVL synthesis problem is described in Section 4. Section 5 presents the experiments conducted, results obtained, and a related discussion and section 6 includes some concluding remarks.

2. Background

An n -variable r -valued function, $f(X)$, is defined as a mapping $f: R^n \rightarrow R$, where $R = \{0,1,\ldots,r-1\}$ is a set of r-logic values with $r \geq 2$ and $X = \{x_1, x_2, \ldots, x_n\}$ is a set of n r-valued variables.

Definition 1: A *min* (minimum) operator is defined as:
$\min(a_1,\cdots,a_n) = a_1 \bullet \cdots \bullet a_n$, where $a_i \in R$.

Definition 2: A *tsum* (truncated sum) operator is defined as:
$tsum(a_1,\cdots,a_n) = a_1 \oplus \cdots \oplus a_n$
$= \min(a_1 + \cdots + a_n, r-1)$, where $a_i \in R$.

Definition 3: A *window literal* of an MVL variable x is defined as:

$$ {}^{a}x^{b} = \begin{cases} (r-1) & if\ (a \leq x \leq b) \\ 0 & otherwise \end{cases} $$

where $a,b \in R$ and $a \leq b$.

Definition 4: A complement of an r-valued variable, l, is defined as:

$$\bar{l} = (r-1) - l$$

Definition 5: A *product term* (PT), $P(x_1,\ldots,x_n)$, is defined as the minimum of a set of *windows literals* on variables $x_1,\ldots,x_n$, i.e. $P(x_1,\ldots,x_n) = c \cdot {}^{a_1}x_1{}^{b_1} \cdot \ldots \cdot {}^{a_n}x_n{}^{b_n}$
$= \min(c, {}^{a_1}x_1{}^{b_1}, \ldots, {}^{a_n}x_n{}^{b_n})$ where $a_i, b_i \in R$, $a_i \leq b_i$ and $c \in \{1, 2, \ldots, r-1\}$.
It should be noted that c is called the value of the PT.

Definition 6: For an MVL function f $(x_1,\ldots,x_n)$, an assignment of values to variables $x_1 = a_1,\ldots,x_n = a_n$ is called a *minterm*, iff: $f(a_1,\ldots,a_n) \neq 0$, where $a_i \in \{0,1,\ldots,r-1\}$.

A *minterm* is a special case of a product term consisting of *literal* and *min* operators where the PT is dependent on all variables and $a_1 = b_1,\ldots,a_n = b_n$. The value of the PT is referred as the value of the *minterm*. If the value of the *minterm* is equal to r, then it is considered as *don't care* and is represented as d. Consider, for

example, the 4-valued 2-variable function shown in Fig. 1. Some of the minterms are $1\bullet^3X1^3\bullet^0X2^0$, $2\bullet^1X_1{}^1\bullet^0X_2{}^0$ and $3\bullet^2X_1{}^2\bullet^1X_2{}^1$.

Definition 7: An implicant of a function $f(x_1,...,x_n)$, is a PT, $I(x_1,...,x_n)$, such that $f(x_1,...,x_n) \geq I(x_1,...,x_n)$ for all assignments of x_i's. In Fig.1, $3\bullet^2X_1{}^2\bullet^0X_2{}^1$ and $2\bullet^1X_1{}^2\bullet^0X_2{}^1$ are examples for implicants.

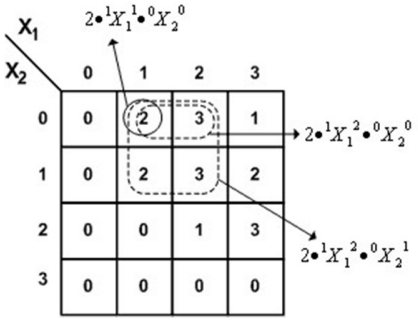

Fig. 1 A tabular representation of f(x1,x2).

A functionally complete set of operators is the set capable of realizing all possible functions. A number of functionally complete sets of operators have been used in the literature. In this paper, we use the set consisting of {*Literal*, *MIN*, *TSUM*, and *Constant*}.

Direct cover (DC) algorithms have been used effectively for synthesis of MVL functions for 2-level programmable logic arrays (PLAs) [1]-[6]. Direct cover approaches for synthesis of MVL functions consist of the following main steps:

1. Choose a Minterm,
2. Identify a suitable implicant that covers that minterm,
3. Obtain a reduced function by removing the identified implicant,
4. Repeat Steps 1 to 3 until no more minterms remain uncovered.

The DC approaches reported in the literature differ in the way minterms are chosen and the way according to which implicants are identified. The algorithm due to [5] selects minterms randomly and selects the implicant resulting in the largest number of zero minterms. The algorithm due to [1] uses the *isolation weight* (IW) for selecting minterms and selects the minimum cost implicant to cover each minterm. The algorithm due to [2] introduces the *isolation factor* (IF) for selecting minterms and selects implicant having minimum Relative Break Count. The last two techniques select minterms in increasing order of values, i.e., start with lower minterm. This called as *considering minterm value.* Consider, for example, the function shown in Fig. 2. The list of minterms and implicants selected by a DC based algorithm is likely to be similar to the one shown in TABLE 1. And the function can be expressed as $F=1\bullet^1X_1{}^2\bullet^1X_2{}^2\oplus 2\bullet^0X_1{}^1\bullet^2X_2{}^3$.

The selection of minterms and the implicants covering them play an important role in obtaining less expensive solutions in terms of the number of product terms required to cover a given function.

3. SYNTHESIS OF Irreversible Ternary MVL Circuit USING PSO

The synthesis of a MVL function can be clarified as a procedure to select implicants that blankets all minterms of the given MVL function. Each Implicant cover more than one minterm. Each minterm might be implicant hence, the amount of implicants that cover all minterm ought to be less than the amount of minterm of minterm itself. However, there exist few cases on which the amount of implicant required to represent to the capacities is equivalent to the minterm.

In addition to that, each minterm itself can be covered by more than one implicants. Consider the example shown in Fig2. Minterm $3^1x_1{}^{12}x_2{}^{20}x_3{}^0$ is covered by both implicants however, DC algorithms select minterm and implicants covering them consecutively until all minterms or of the given function are covered. Thus, the order of selected minterm is very important. Every DC algorithm proposes diverse criteria in selecting minterm (and implicants) and none in the event that they claim to produce the minimum number of product term. In addition to that, when a minterm is chosen, the implicant having the same worth if that minterm will be chosen. Thus, if somehow minterm $3^1x_1{}^1\bullet^2x_2{}^2\bullet^0x_3{}^0$in Fig. 2 is selected by a DC algorithm, the implicant that can be selected by the algorithm should have the value of 3 which is $3^1x_1{}^1\bullet^2x_2{}^2\bullet^0x_3{}^0$. However, this implicant will not simplify the function. In fact, choosing implicant $3^1x_1{}^1\bullet^2x_2{}^2\bullet^0x_3{}^0$requires us to select 3 more implicants to cover all minterm of the function.

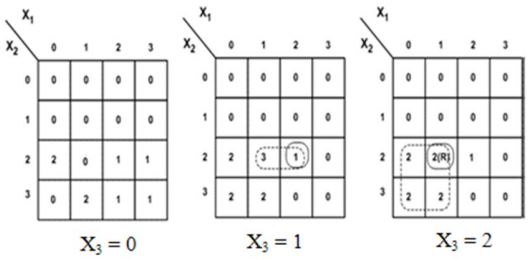

Fig. 2 An example of DC algorithm.

3.1 Particle Representation:

In this paper, the synthesis of a given MVL function is modeled as a mapping between minterm and implicant covering it. The goal is to select least number of implicants to represent the given function. Fig. 3 shows the mapping between minterm and implicant from the example given in Fig. 2. In this Figure, the bold line depicted fully covered by relationship while the line indicates partially covered by relationship.

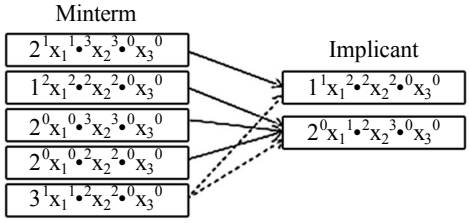

Fig. 3. Minterm and implicant mapping for example in Fig. 2.

The truth table of the given MVL function will be represented as a series of integers. The length of this string is equal to r^n where n is the number of variables. For 3-variable 3-valued function, the length of truth table is then equal to 27. Thus, the example show in Fig. 2 can be expressed by integer string: "000000002011021100000000231022000000000022102200".

Each particle in the swarm consists of 96 elements include 2 part of 48 element that are represent F1 and F2.Inthis paper, outputs F1 and F2 have been synthesized in order to further understand theonlyF1isshownin all Figures. Each of these element consist of 7 integer attributes representing implicant covering the corresponding minterm, i.e., the first element represent the (selected) implicant covering the first minterm and so on. The first attributes is for the value of the implicant, the second and third are for the window of the first variable while the next two attributes are for the window of the second variable and the last two attributes are for the window of the third variable. Thus, implicant $2^0x_1^1\bullet^2x_2^3\bullet^3x_3^1$ is expressed as 2012331. Unlike DC algorithm we will consider all implicants that can cover a minterm, whether the implicant is completely a partially covering the minterm. Table 1shows all implicants covering minterm $3^1x_1^1\bullet^2x_2^2\bullet^0x_3^0$ for the example shown in Fig. 2. In this table, the last implicant (3112200) is the only implicant that is fully covering minterm 3112200.

Furthermore, Fig. 3 shows that implicant $2^0x_1^1\bullet^2x_23\bullet^0x_3^0$ covers the following minterms: $2^0x_1^0\bullet^2x_2^2\bullet^0x_3^0$, $2^0x_1^0\bullet^3x_2^3\bullet^0x_3^0$, $2^1x_1^1\bullet^3x_2^3\bullet^0x_3^0$ and $3^1x_1^1\bullet^2x_2^2\bullet^0x_3^0$. According to this Figure, those minters have to be mapped to implicant $2^0x_1^1\bullet^2x_2^{30}x_3^0$. However, assuming that we permit more than one minterm to be mapped to the same implicant, each time an implicant is chosen, we have to check if this implicant is mapped by other minterm or not. We accept that this methodology will confine the power of evolutionary process of swarm intelligence in addition to extra computation in comparing selected implicants. However, the minterm that are already covered by an implicant should be somehow mapped to an implicant that will be ignored at the final representation. For this purpose we introduce a special implicant 0000000. This implicant is an additional implicant for each minterm to select. Thus, the list shown in table 1 should be amended by this minterm.

Table 1: ALL IMPLICANT COVERING MINTERM 3112200

3112200
1112200
1122200
1012200
1022200
1112300
2112200
2112300
2012300
3112200

Assume that there are three particles in swarm whose best representation shown Fig. 4. We can see here, some of the minterm in particle 1 and 2 mapped to implicant '0000000'. Indeed, particle 1 and particle 2 yield to the same functional representation of the function. However, each minterm in particle 3 is mapped to a non-'0000000' implicant. The final representation of the function of particle 3 is then equal to the TSUM of all of its selected implicants, i.e., which is not correct.

Table	Partice1	Particle2	Particle3
0	0000000	0000000	0000000
0	0000000	0000000	0000000
0	0000000	0000000	0000000
0	0000000	0000000	0000000
0	0000000	0000000	0000000
0	0000000	0000000	0000000
0	0000000	0000000	0000000
0	0000000	0000000	0000000
2	2012300	0000000	2012300
3	0000000	1122200	2012300
1	1122200	0000000	1122200
0	0000000	0000000	0000000
2	0000000	2012300	2012300
2	0000000	0000000	2012300
0	0000000	0000000	0000000
0	0000000	0000000	0000000

Fig. 3. Example of particle representation.

3.2 Particle Fitness

The fitness function is separated into two parts. The first part is called *Functional Fitness*, F_f. the second part of the fitness function is called *Objective Fitness,* F_o. Functional fitness is obtained by comparing the truth table of the given function with that of obtained one. It is equal to the number of minterm matching (hits), N_h, between the two truth tables less the number of mismatch, N_m, between the two truth tables. This can be formulated as $F_f = F_o + N_h$. with N_p being the number of product terms, the Objective Fitness is formulated as $F_o = (100\text{-}N_p)/100$.

The overall fitness is then calculated as the sum of functional fitness with objective fitness. Thus, a solution with the highest functional fitness, then the value of objective fitness will determine which of these to use as the best result.

TABLE 2: FITNESS CALCULATION FOR PARTICLE SHOWN IN Fig.4

P	Truth Table Obtained	N_h	N_m	F_f	N_p	F_o	Fit
1	0000000033102200000000000023102200000000023102200	47	1	46	6	0.94	46.94
2	0000000023102200000000000023102200000000023102200	48	0	48	6	0.94	46.94
3	0000000023102200000000000023102200000000023102200	48	0	48	4	0.96	48.96

Table 2 shows the truth table obtained by each of this particle and its fitness respectively. The table shows that there is a miss in truth table matching for particle 1, which means that this particle is not representing the given MVL function correctly. On the other hand, both particle 2 and 3 represent the given MVL function correctly. However, particle 3 use less number of implicant compared to partcle2. Thus the objective function if particle 3 is bigger than that of particle 2, which result in higher overall fitness valued. As can be seen in the table, particle 3 has the best fitness and considered the best solution for the given MVL function.

3.3 Particle Movement

In the initialization step, each element of the particle is loaded with randomly chosen implicant. We can consider velocity as the number of displacement from current position to global best. For example, if the particle is moving with velocity 1 at position 2 and the position of best particle was 1 and position of global best was 0, assuming that all coefficients are equal to 1, the new velocity for the particle will be equal to 1 + (1-2) + (0-2) = -2. In discrete nature of domain, velocity is equal with the displacement and each element of displacement should be an integer value. The displacement of a particle is calculated as followed:

$$D_{t+1,i} = c1\ D_{t,i} + DS_{t,i} + DG_{t,i}$$

$$DS_{t,i} = c2r2(P_{i,t} - X)$$

$$DG_{t,i} = c3r3(P_{g,t} - X)$$

Where

$D_{t,i}$=displacement of particle i at time step t

$X_{t,i}$=position of particle i at time step t

$DS_{t,i}$ = Displacement of particle I at time step t due to its best experience

$DG_{t,i}$ = Displacement of particle I at time step t due to its global best experience

Pi,t= previous best position at time step t

Pg,t = previous global best position at time step t

C1m c2, and c3 = social/cognitive confidence coefficients

R2 and r3 = random numbers

The calculation of $D_{t+1,\,i}$ is performed just to find out whether the amount of moves exceed V_{max} or not.

To give a superior picture of the methodology, assume that the position of a particle at time t is shown as particle 1 in Fig. 4. While the global best position as particle 2 in Fig. 4, and best position is shown as particle 3. Assume also that the displacement at time t was 2, V_{max} is equal to 5 and all random number and coefficients are equal to 1.

Possible move to Best particle	Possible move to Global Best particle
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	2012200 -> 2012300
0	0000000 -> 1122200
0	1122200 -> 0000000
2012300 -> 2003300	0
0000000 -> 2113300	2012300 -> 0000000
0	0
0	0

Fig. 4 Possible move for a particle at time *t*

Fig. 5 shows the possible moves for the particle. These moves are the illustration of DS and DG. Thus, the value of DS is equal to 2 and DG id equal to 4. The value of $D_{t+1,\,i}$ is equal to 2+2+4 = 8. Since the value of $D_{t+1,\,i}$ is larger than V_{max}, we need to Scale down the value of DS and DG. The value of DS will then equal to $(2/8)*V_{max}$ =1.25. Since we can only have an integer value, the value

of DS is then equal to 1. Similarity, the value of DG is now equal to (4/8)*V_{max} = 2.5≈ 2. Using the new values, we can now perform the moves. Since Ds is equal to 1, we now need to randomly select one of the tow possible moves (to best Particle) shown in Fig. 5. Similarity, we will then need to randomly select tow out of four possible moves (to global best) shown in Fig. 5. Since DS+DG is still smaller than $D_{t+1, i}$, we will then need to perform $D_{t+1, i}$–(DS+DG) random move to the particle, to d this, can selected any minterm randomly and then select an implicant covering it randomly.

4. Heuristics for Irreversible Ternary MVL Circuit

As indicated in [7] and [8], only 4-valued logic with 2-inputs circuits has been mentioned. We extend and improve this by 3-valued logic and 3-inputs.

Another trick that we used for better result of synthesis is to synthesis the all function results (ex. F_1 and F_2) in a same time, because for synthesizing irreversible circuits as reversible circuits, that has two outputs, we need to synthesis both output simultaneously.

For better result, we change definition of tsum according to using reversible gates in irreversible circuits, then:

TSUM (minterm1, minterm2) =
mod (value_minterm1 + value_minterm2, 3)

In other words, this is concept of add in GF3.

GF3 (a,b,c, +) = (a+b+c mod3)

5. EXPERIMENTAL RESULTS AND COMPARISON

We synthesis both irreversible and reversible circuits (listed in Table 3) using our proposed heuristics. Parameters used in our algorithm are shown in Table 4.

The result shows that this method is useful to synthesis and improves irreversible MVL circuits. As shown in Table 5, tested circuits in our heuristics has less product terms versus [7] and [8].

Table3: Tested Circuits

Function	Input / Output
Prod*n*	$x_0\ x_1 \cdots x_{n-1}$
	$y = (x_0 x_1 \cdots x_{n-1}) \bmod 3$
sumn	$x_0\ x_1 \cdots x_{n-1}$
	$y = (x_0 + x_2 + \cdots + x_n) \bmod 3$
ncyr	$x_0\ x_1 \cdots x_{n-1}$
	$y = \left[\sum_{i=0}^{n-1} + \left(\prod_{j=0}^{r-1} x_{(i+j) \bmod n}\right)\right] \bmod 3$
Sqsumn	$x_0\ x_1 \cdots x_{n-1}$
	$y = \left(x_0^2 + x_1^2 + \cdots + x_{n-1}^2\right) \bmod 3$
Avgn	$x_0\ x_1 \cdots x_{n-1}$
	$y = \text{int}\left[(x_0 + x_1 + \cdots + x_{n-1}) / n\right] \bmod 3$
a2bcc	$a\ b\ c$
	$y = (a^2 + bc + c) \bmod 3$
Thadd	$a\ b$
	$c = \text{int}\left[(a+b)/3\right],\ s = (a+b) \bmod 3$
tfadd	$a\ b\ c$
	$y = \text{int}\left[(a+b+c)/3\right],\ s = (a+b+c) \bmod 3$
mul2	$a\ b$
	$c = \text{int}\left[ab/3\right],\ m = ab \bmod 3$
mul3	$a\ b\ c$
	$c = \text{int}\left[abc/3\right],\ m = abc \bmod 3$

Table 4: Parameters of Algorithm

max_iterations	100
no_of_particles	20
g_best_fitness	0
c1	1
c2	0.5
c3	0.5
Vmax	5

Table 5: Comparison Results

Function	Number of minterms	Number of PTs [7][8]	Number of PTs
Prod*n* (Pruduct n)	8	8	4
Sum*n* (Sum n)	18	14	12
Ncyr (n cyclic r)	16	13	8
Sqsum*n* (Square sum n)	21	15	9
Avg*n* (Average n)	18	13	5
a2bcc	18	14	7
Thadd (Ternary half adder)	F1=S=6, F2=c=3	7	6

Tfadd (Ternary full adder)	F1=S=18, F2=c=11	25	19
mul2 (multiple 2)	F1=M=4, F2=c=1	(M=4+c=1)=5	3
Mul3 (multiple 3)	F1=M=8, F2=c=4	(m=8+c=4)=12	7

Length of function is important in PSO algorithm. Our method cans synthesis 3-valued 3-inputs, that were 96*7-bit input for PSO, in acceptable and admissible time. As seen in table 1, our heuristics lead to less product terms and that is equal to less gate and less gate consumes less power and less waste power is one of synthesis goals.

6. Conclusions

Some heuristics on discrete Particle Swarm Optimization based technique for synthesis of MVL functions is presented in this paper. The comparison made with the other algorithms shows that the technique introduced produces better results, in terms of the average number product terms needed to synthesize a 3-valued 3-input given MVL function.

References

[1] Besslich, Philipp W. "Heuristic minimization of MVL functions: a direct cover approach." Computers, IEEE Transactions on 100, no. 2 (1986): 134-144.

[2] Yildirim, Cem, Jon T. Butler, and Chyan Yang. "Multiple-valued PLA minimization by concurrent multiple and mixed simulated annealing." In Multiple-Valued Logic, 1993., Proceedings of The Twenty-Third International Symposium on, pp. 17-23. IEEE, 1993.

[3] Yang, Chyan, and Y-M. Wang. "A neighborhood decoupling algorithm for truncated sum minimization." In Multiple-Valued Logic, 1990., Proceedings of the Twentieth International Symposium on, pp. 153-160. IEEE, 1990.

[4] Pomper, Gardner, and James R. Armstrong. "Representation of multivalued functions using the direct cover method." Computers, IEEE Transactions on 100, no. 9 (1981): 674-679.

[5] Dueck, G. W., and D. M. Miller. "Direct cover MVL minimization using the truncated sum." In Proceedings of the International Symposium on Multiple-Valued Logic. IEEE, 1987.

[6] Tirumalai, Parthasarathy P., and Jon T. Butler. "Minimization algorithms for multiple-valued programmable logic arrays." Computers, IEEE Transactions on 40, no. 2 (1991): 167-177.

[7] Sarif, Bambang AB, and Mostafa Abd-El-Barr. "Functional synthesis using discrete particle swarm optimization." In Swarm Intelligence Symposium, 2008. SIS 2008. IEEE, pp. 1-8. IEEE, 2008.

[8] Abd-El-Barr, Mostafa. "Ant Colony Heuristic Algorithm for Multi-Level Synthesis of Multiple-Valued Logic Functions." IAENG International Journal of Computer Science 37, no. 1 (2010).

[9] Kalganova, T., J. Miller, and N. Lipnitskaya. "Multiple-valued combinational circuits synthesized using evolvable hardware approach." In Proc. of the 7th Workshop on Post-Binary Ultra Large Scale Integration Systems (ULSI'98) in association with ISMVL, vol. 98. 1998.

[10] Moraga, Claudio, and Wenjun Wang. "Evolutionary Methods in the Design of Quaternery Digital Circuits." In PROCEEDINGS OF THE INTERNATIONAL SYMPOSIUM ON MULTIPLE VALUED LOGIC, vol. 28, pp. 89-94. INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, 1998.

[11] Sarif, Bambang AB, and Mostafa Abd-El-Barr. "Synthesis of MVL Functions-Part I: The Genetic Algorithm Approach." In Microelectronics, 2006. ICM'06. International Conference on, pp. 154-157. IEEE, 2006.

[12] Abd-El-Barr, Mostafa, and Bambang AB Sarif. "Synthesis of MVL Functions-Part II: The Ant Colony Optimization Approach." In Microelectronics, 2006. ICM'06. International Conference on, pp. 158-161. IEEE, 2006.

[13] Abd-El-Barr, Mostafa, and Bambang AB Sarif. "Improved Ant Colony Optimization Approach for Synthesis of MVL Functions." In 2007 International Conference on Instrumentation, Communication, and Information Technology. ICICI, pp. 8-9. 2007.