# Minimization of Incompletely Specified Mixed Polarity Reed Muller Functions using Genetic Algorithm

B. A. AL JASSANI
b.a-aljassani@napier.ac.uk

N. URQUHART
n.urquhart@napier.ac.uk

A.E.A. ALMAINI
a.almaini@napier.ac.uk

Edinburgh Napier University
Edinburgh, UK

***Abstract* - A New and efficient Genetic Algorithm (GA) based approach is presented to minimise the number of terms of Mixed Polarity Reed Muller (MPRM) single and multi output incompletely specified Boolean functions. The algorithm determines the allocation of don't care terms for the given function resulting in optimal MPRM expansions. For an *n*-variable function with β unspecified minterms there are ($3^n \times 2^{\beta}$) distinct MPRM expansions. A minimum MPRM is one with the fewest products. The algorithm is implemented in C++ and fully tested using standard benchmark examples. For the benchmark examples tested, the number of terms is reduced, on average, by 49% if "don't care" terms are included.**

***Keyword- Mixed Polarity Reed Muller, Incompletely specified Boolean functions, Genetic Algorithm***

## I. INTRODUCTION

The Mixed Polarity Reed Muller (MPRM) expressions are one of the canonical AND_ExOR expressions [1] as shown in (1).

$$F(x_{n-1}x_{n-2}\ldots x_0) = \bigoplus_{i=0}^{2^n-1} b_i P_i \qquad (1)$$

Where $\oplus$ represents ExOR logic gate, $\sum$ is the sum operator, and $P_i$ are the product terms of the Reed-Muller function, and $i=0,1,2,\ldots 2^n-1$.

For completely specified Boolean functions, $b_i \in \{0,1\}$ and $b_i = 1$ indicates the presence of the corresponding terms in the expansion.

$$P_i = \prod^{i_k} x_k = x_{n-1}^{i_{n-1}} x_{n-2}^{i_{n-2}} \ldots\ldots x_0^{i_0}$$

Where

$$x_k = \begin{cases} 1 \text{ or } x_k & if \quad i_k = 0 \\ 1 \text{ or } \bar{x}_k & if \quad i_k = 1 \\ \bar{x}_k \text{ or } x_k & if \quad i_k = 2 \end{cases}$$

Where $i_k$ refers to the polarity of each varaible of the functions.
In MPRM, each variable can appear as true, complemented or both at the same time. There are $3^n$ sets of MPRM expansions. Each expression can be identified by a polarity number. The polarity of MPRM expansion can be represented by replacing each variable by 0, 1, or 2 depending on whether the variable is used in true, complement or mixed respectively. The polarity will be the decimal equivalent of the resulting ternary number.

Example (1) The Polarity 7 for the 3 variable function $f(x, y, z) = \sum (0, 2, 6, 7)$ is as follow:

In $Polarity\ (021)_3 = Polarity\ (7)_{10}$,
$f(x, y, z) = \bar{y}\,\bar{z} \oplus y\,\bar{z} \oplus x\,\bar{y}\,\bar{z} \oplus x\,y \oplus x\,y\,\bar{z}$
Variable *x* appears in true form; variable *y* appears in mixed form while variable *z* appears in the complement form.
The problem here is how to find efficient solutions amongst the very large number of polarities in the MPRM domains for incompletely specified Boolean functions without resorting to exhaustive search.

Many authors have considered the problem of finding the optimal Reed Muller (RM) expansions with the least number of terms. GREEN [2,3] described the set of $3^n$ consistent MPRM canonical forms of an n-variable switching function and investigated the structures of the various fixed and mixed polarity transforms. McKENZIE, *et al.* [4] presented a non exhaustive techniques to determine the allocation of don't care terms for incompletely specified Boolean functions resulting in reduced Fixed Polarity Reed Muller (FPRM) expansions.

HABIB[5,6] proposed a new procedure to generate FPRM for completely and incompletely specified functions. He also presented another new technique to generate minimal MPRM expansions for completely and incompletely specified Boolean functions. The author tested his own methods with random functions of up to 10 variables.

HELLIWELL and PERKOWSKI [7] attempted to minimize multi output completely and incompletely specified Boolean functions to find the minimal MPRM by using heuristic methods. The algorithm named "xlinking" is based on a new cube operation that generalizes known operations of merger, exclusion and other logic operations and tested his own method with random functions of up to 19 variables.

WANG and ALMAINI [8] presented a new technique to obtain the best polarity of FPRM expressions for large multiple output Boolean functions.

There are other interesting aspects of logic design such as multi-valued and multi-levels, especially the graphical approach based on Binary Decision Diagram (BDD) and RM_BDD as in [9, 10,11]. These, however, are outside the scope of this paper.

This paper introduces a new strategy for the optimization of incompletely specified Boolean functions resulting in minimum MPRM expansions using Genetic Algorithm (GA). The GA splits into two different stages to improve the performance of the algorithm and to reduce the computation time. The first stage conducts a polarity search to find the best MPRM expansions with fewer terms without considering the "don't care" terms. In the second stage, the "don't care" terms are used to further minimise the expressions. The rest of the paper is organized as follows. Section II gives a review of MPRM for incompletely specified functions. An algorithm utilising a GA to find optimal MPRM among $3^n$ different polarities for incompletely specified multi output Boolean functions is proposed in Section III. Section IV gives the definitions and operation of the GA. Section V shows the experimental Results. Conclusions are presented in Section VI.

## II. REVIEW OF MPRM EXPANSIONS FOR INCOMPLETELY SPECIFIED FUNCTIONS

An incompletely specified Boolean function is a function with one or more minterms with undefined values. These unspecified minterms are known as "don't care" terms and sometimes can help the process of minimization. Any n_variable Boolean logic function may be represented in a sum of products form as:

$$F(x_{n-1}x_{n-2}\ldots x_0) = \sum_{i=0}^{2^n-1} a_i m_i + \sum_{i=0}^{2^n-1} d_i m_i \qquad (2)$$

Where $m_i$ are the minterms; $a_i$ and $d_i \in \{0,1\}$ are coefficients which may take the value 0 or 1, $a_i = 1$ indicates the presence of minterms, and $d_i = 1$ indicates the presence of don't care terms.

When incompletely specified Boolean functions are transformed to the RM domain, "don't care" terms transform along with the specified terms and their effect is distributed over several terms of the new representation. Therefore, it is necessary to find an optimum selection of these terms to minimize the number of terms in the expressions.

For incompletely specified functions of the type given in (1), $b_i \in \{0,1, DC\}$. However, when the value of $b_i$ is undefined, it may take the value 0 or 1 without effecting any change to the output of the function. These products $P_i$ are unspecified or "don't care" terms for the given function. Then the RM expansion may be denoted an incompletely specified RM expansion.

For completely specified Boolean functions of n-input variables, there exist $3^n$ MPRM expansions with different number of terms. For incompletely specified function, the numbers of MPRMs increases exponentially with the increase of the number of "don't care" terms. There are $(3^n \times 2^\beta)$ distinct MPRMs for n-variable functions with β "don't care" terms. An expression with the fewest products is an optimum expression.

## III. A GENETIC ALGORITHM FOR INCOMPLETELY SPECIFIED MULTI OUTPUT FUNCTIONS

The previous GA based approach for solving single output incompletely specified Boolean functions [12] was found to take excessive time when evolving a solution. In order to address the performance issues the approach presented for solving multi output incompletely specified functions is to split the GA developed previously into two interrelated stages. The aim is to deduce the optimal selection of "don't care" values to ensure the minimal Reed Muller expansion.

The GA-based approach presented by the authors for multi output incompletely specified Boolean functions utilises two populations. The first to represent the mixed polarity and the second to indicate the presence or absence of "don't care" terms. The first population is evolved, the genotypes from the final population being used to seed the second population which is then evolved to produce a final solution. The results show that the algorithm can produce good result for small functions with small number of don't care terms. For functions with large number of variables and " don't cares ", the number of possible solutions is $(3^n \times 2^\beta)$ requiring excessive CPU time. To overcome this problem, a two stage GA is proposed. The proposed GA has two search spaces depending on two different variables (n & β). The 1st search space is $3^n$ while the other is $2^\beta$. The GA splits into two stages to cope with large functions. The first stage is to produce the best MPRM expansions without including "don't care" terms. The number of individuals resulting from the first stage equals to the population size which is one of the GA parameters entered by the user to run the algorithm. The aim of this stage is to minimise the search space saving computation time. The second stage will use the resulting best individuals from the first stage with different collections of "don't care" terms to further simplify the expressions. The pseudo code of the proposed GA is shown in Fig.1.

## IV DEFINITIONS

### *A. Population*

The GA algorithm starts with a set of solutions called

```
GEN_NO.  Number of generations
Pop_size     size of the populations
tnsize       Tournament size
Begin ( )
{
  Input GA-parameter
  Read the given function in Boolean domain
  Randomly initialize the population for the polarity
  Randomly initialize the population for the don't care
Loop for (GEN_NO/3)
  {
  Select one parent from the polarity population .
  Select another parent from the polarity population .
  Make Single Point Crossover to produce Child1.
  Make Uniform Mutation on the Child1.
  Fitness to find number of terms for the Child1
  without including "don't care" terms.
  If ((Child1_fit < fitness of parents)  And
                        (Child1_fit < >any current fitness)) .
      Do Replacement.
  }
If there is a don't care terms for the input function
  {
Loop for  (2*GEN_NO/3)
    {
    Select parent from the "don't care" population.
    Select another parent from the "don't care" population.
    Make Single Point Crossover to produce Child2.
    Make Uniform Mutation on the Child2.
    Select one parent from the polarity population.
    Make Uniform Mutation on it to produce Child1.
    Fitness to find number of terms for Child1 including
    the selected don't care terms specified in Child2.
   If (Resulted fitness < fitness of parents)
       Do Replacement for the Child1.
       Do Replacement for the Child2.
    }
  }
}
```

Figure 1. Pseudo code of GA

population. Each feasible solution in population is called individual. Each individual is a sequence of genes. The proposed algorithm contains two populations each with their own representation. The first population is represented using ternary numbers to hold the polarity number of the MPRM. The size of the polarity population equals n-bits for n-variable functions. The second population is represented using binary numbers to indicate the presence or absence of "don't care" terms. The size of the population for the "don't care" equals to the total number of "don't care" terms for all outputs for the given functions.

**Example (2):** Consider a 5-variable function with 3 outputs and 5 "don't care" terms for each output . The individuals in both populations will be as shown in Figure (2)

Bits (0 – 4) in Fig. (2a) contain ternary number to indicate polarity number $(12112)_3 = (149)_{10}$

Bits(0 – 14) in Fig. (2b) contain binary number to indicate the existence of "don't care" terms. This individual indicates the existence of two "don't care" terms for the first output, three "don't care" terms for the 2nd output, and one "don't care" term for the 3rd one.

*B. Fitness Function*

Fitness function is implemented to convert the specified Boolean function to the RM domain and computes the number of terms for the polarity specified in the polarity population. Initially it computes the fitness of all the chromosomes and then it computes the fitness for the new offspring of each evaluation. The fitness function is implemented by using a new algorithm based on tabular techniques in [12].

As outlined above, the proposed GA has two stages and therefore two fitness functions: The 1st fitness function uses the method in [12] to convert the specified Boolean function to the RM domain and find the number of terms for the polarity represented by the polarity population without considering the "don't care" terms. The 2nd fitness function uses the same method in [12] to calculate the number of terms for the individual being evaluated. Each new member of the population is based upon individuals selected from the first stage population being combined with the second stage population.

*C. Selection*

Tournament selection with a tournament size *t* is used throughout, where *t* is specified by the user.

*D. Crossover*

Crossover is the principle genetic operator. It operates by selecting two individuals randomly (tournament selection) and generates one offspring. The child inherits some of

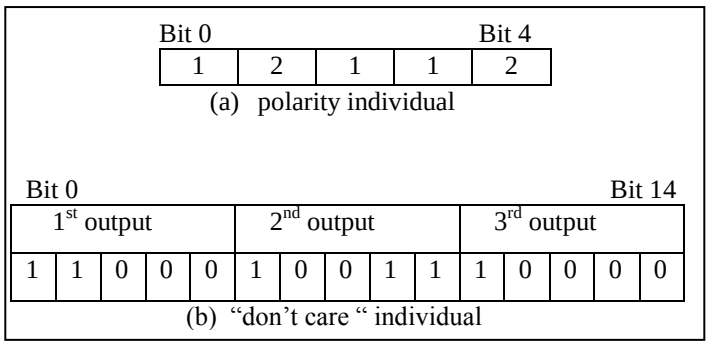


Figure 2. Details of individuals for example (2)

the chromosomes from one parent and the rest from the other parent. The crossover operation is the action of choosing randomly a crossover point and combining two different parts from the two parents to form a new offspring. This method is called a single point crossover. For incompletely specified Boolean functions, the crossover will be carried out twice. Once for each population.

*E. Mutation*

The mutation operator alters a single gene of the individual randomly. It is carried out by selecting one bit at random and replacing the value held in the selected bit with random ternary number. This type of mutation is called uniform mutation. The mutation will also be done twice for incompletely specified Boolean functions.

*F. Replacement*

Tournament replacement method is used in the proposed GA. The algorithm randomly chooses a number of individuals (equal to determined tournament size) from the population and replaces the loser which has the worse fitness with the new offspring generated. For incompletely specified functions, the replacement will be done for the two populations. To avoid premature convergence, the algorithm will not replace the new chromosome if there is another individual with the same fitness in the current populations.

**Example (3)** Consider the incompletely specified Boolean function with three inputs and two outputs as shown in Table I.

It is clear that this function can be represented using polarity population with 3 bit and "don't care" population with 5 bit. The populations are used to represent mixed polarity/"don't care" terms using ternary/binary code respectively.

1) Assuming that the user specifies population size= 7. Then, 7 individuals to represent polarity are initialized randomly using ternary code as detailed in Table II.
2) Another 7 individuals to represent "dont care" selection are initialized randomly using binary numbers as shown in Table III.

TABLE I. Truth table for the given Boolean function

| **Inputs** | **Outputs** | |
|---|---|---|
| $X_1X_2X_3$ | $Y_1$ | $Y_2$ |
| 000 | 0 | 1 |
| 001 | 1 | DC |
| 010 | 0 | 1 |
| 011 | DC | 0 |
| 100 | 1 | 1 |
| 101 | DC | 0 |
| 110 | 1 | DC |
| 111 | DC | 1 |

TABLE II. Initialization of the polarity population

| **Polarity Population** | | | **Polarity number** |
|---|---|---|---|
| ***Bit 2*** | ***Bit 1*** | ***Bit 0*** | ***(Decimal)*** |
| 2 | 1 | 1 | 22 |
| 0 | 1 | 1 | 4 |
| 1 | 2 | 0 | 15 |
| 1 | 2 | 2 | 17 |
| 1 | 0 | 2 | 11 |
| 2 | 1 | 0 | 21 |
| 1 | 2 | 1 | 16 |

TABLE III. Initialization of the "don't care" population

| **ID** | **"Dont care" population** | | | | | **Information** |
|---|---|---|---|---|---|---|
| | ***Bit 4*** | ***Bit 3*** | ***Bit 2*** | ***Bit 1*** | ***Bit 0*** | |
| 1 | 1 | 0 | 1 | 0 | 1 | The 1st and 3rd don't care for the first output exist and the 2nd don't care for the second output exists. |
| 2 | 0 | 1 | 1 | 1 | 0 | The 2nd and 3rd don't care for the first output exist and the 1st don't care for the second output exists. |
| 3 | 0 | 1 | 1 | 0 | 1 | The 2nd and 3rd don't care for the first output exist and the 2nd don't care for the second output exists |
| 4 | 1 | 0 | 0 | 1 | 0 | The 1st don't care for the first output exists and the 1st don't care for the second output exists. |
| 5 | 1 | 1 | 1 | 0 | 1 | All the don't care for the first output exist and the 2nd don't care for the second output exists. |
| 6 | 1 | 1 | 0 | 1 | 1 | The 1st and 2nd don't care for the first output exist and all the don't care terms for the second output exist. |
| 7 | 0 | 0 | 0 | 1 | 0 | Only the 1st don't care term for the second output exists. |

Consequently, we have $3^3$ possible polarities and for each one of these polarities there are $2^5$ possible collections of " don't care " terms. Hence, there are ($3^3 \times 2^5 = 864$) possible solutions. There is different truth vector for each output for the different ID's detailed in Table III as explained in Table IV.

TABLE IV Truth vector for each output

| ID | Truth vector of the 1st output for the given function | Truth vector of the 2nd output for the given function |
|---|---|---|
| 1 | 01011011 | 10101011 |
| 2 | 01001111 | 11101001 |
| 3 | 01001111 | 10101011 |
| 4 | 01011010 | 11101001 |
| 5 | 01011111 | 10101011 |
| 6 | 01011110 | 11101011 |
| 7 | 01001010 | 11101001 |

3) GA splits into two stages to improve its performance. The GA selects two parents from the populations using tournament selection. The crossover is then applied to produce new child which has a mutation applied. Finally, it replaces one of the individuals in the population with the new child if the number of terms for the child is less than the number of terms for the parents. This evolution loop of the GA is the same for both stages. The only difference is the fitness function.

4) In the first stage, the algorithm will run for one third of the number of generations determined by the user. It will find the number of terms for each individual from the polarity population without considering the "don't care" terms. The GA will produce the best individuals with less number of terms as specified by the user, 7 in this example. The fitness function uses the method in [12] for converting the function from the Boolean domain to RM domain. When GA is running, the best seven individuals are produced as shown in Table V.

5) In the second stage, the algorithm will run for two third of the number of generations determined by the user. It will add the selected "don't care" terms to the truth table for the given function as explained in Table IV according to the individuals from the "don't care" population in Table III. Then convert the function from the Boolean domain into RM domain and calculate the number of terms for the incompletely specified Boolean function for each output considering the sharing between these outputs.

TABLE V. Best individuals produced from the first stage of the proposed GA

| Polarity number in ternary code | | | Number of terms |
|---|---|---|---|
| 0 | 1 | 2 | 6 |
| 1 | 0 | 1 | 5 |
| 2 | 1 | 2 | 6 |
| 1 | 2 | 0 | 7 |
| 1 | 2 | 1 | 6 |
| 0 | 1 | 0 | 7 |
| 0 | 0 | 1 | 7 |

## V. EXPERIMENTAL RESULTS

The program was applied to several MCNC and LGSynth93 benchmark functions [13, 14].The algorithm was executed on a personal computer with an Intel CPU running at 2.4 GHz and 2 GB RAM under Window XP, professional. The algorithms are implemented using C++. The results of multi stage GA are given in Table VI. Each result from GA is taken after running the GA algorithm ten times. *T* denotes to the number of terms for the given benchmark. Number of "don't care " terms are given in *N.DC. M. terms* gives the minimum number of terms with and without including the "don't care" terms. S denotes percentage saving. Avg./STD denotes to average number of terms for the ten runs of the proposed GA / Standard deviation. STD tells how closely a set of results is clustered around the average of the results. If all the results during 10 runs of the GA are the same, STD equals 0.

The average saving in the number of terms for multi stage GA is 49 % for MPRM using "don't care" compared with MPRM without using "don't care" terms. Although some authors [4,5,6,7] have implemented algorithms for minimization of MPRM for incompletely specified Boolean function, no benchmark results have been published that could be compared to the result in this paper.

By testing a number of examples using single and multi stage GA, it was found that both algorithms have similar results but the time required for the GA with two stages is much less as compared with one stage GA as shown in Table VII. For example, the multi stage GA took 2 seconds to find the optimal polarity for the benchmark example life compared to 16 hours using single stage GA. The other difference between the results of these two algorithms is the standard deviation. The standard deviation and average for the result of the single stage GA are much higher than multi stage GA especially when the function has large "don't care" terms.

TABLE VI. Benchmark results for the multi stage GA

| Name | I/O | T | N. DC | M. Terms / Time (without DC) | M. Terms / Time (with DC) | S % | Avg./ STD |
|---|---|---|---|---|---|---|---|
| Xor5 | 5/1 | 16 | 16 | 6/<1 sec. | 1 /<1 sec. | 83 | 1/ 0 |
| Rd53 | 5/3 | 31 | 54 | 20 <1 sec. | 6 /1 min. | 70 | 10.1/1.3 |
| Rd73 | 7/3 | 127 | 63 | 63 <1 sec. | 63 / 2 sec. | 0 | 63 / 0 |
| Squar5 | 5/8 | 29 | 32 | 26/<1 sec. | 23 /1 sec. | 11 | 24 /1.1 |
| Sym10 | 10/1 | 837 | 187 | 306/1 sec. | 64 / 1 min. | 92 | 86/ 16.7 |
| 9sym | 9/1 | 420 | 92 | 173/1 sec. | 34 / 40 sec. | 72 | 52 /8.9 |
| life | 10/1 | 140 | 372 | 84 1sec. | 40 / 2 sec. | 52 | 44/8 .2 |
| clip | 9/5 | 496 | 80 | 182/8 sec. | 182 /8 sec. | 0 | 182 / 0 |
| newtag | 8/1 | 234 | 22 | 6/1 sec. | 1 / 16 sec. | 66 | 2 / 0.5 |

## VI. CONCLUSIONS

The process of optimization of the MPRM for incompletely specified multi output functions is computationally hard problem, because of the large search space which increases with increasing number of variables and "don't care" terms. The problem is further complicated when different outputs have different "don't care" terms. The proposed GA splits into two stages. The first stage produces best individuals without including "don't care" terms. The second stage deduces the optimal selection for the "don't care" terms which minimize the number of terms for the individuals produced in the first stage.

From the results shown for the GA with single and multi stage, it is clear that multi stage GA is more efficient in terms of time and average of results especially for large number of variables and don't cares. In reality, most functions have a significant number of "don't care" terms (reach 1000 or more), then there are $2^{1000}$ different representations for don't care terms. Therefore, it is recommended using the multi stage GA to reduce the time taken and to produce good results no matter how large the number of "don't care" terms.

The multi stage GA algorithm for incompletely specified functions was tested with benchmark functions and the tests show better results (average saving 49%) are achieved when "don't care" terms are taken into account in the attempted examples.

TABLE VII. Comparison between Single and Multi stage GA

| | | **Multi Stage GA** | | | **Single stage GA [12]** | | |
|---|---|---|---|---|---|---|---|
| **Name** | **N. DC** | **M. Term** | **Time** | **Avg/ STD** | **M. Term** | **Time** | **Avg/ STD** |
| Xor5 | 16 | 1 | 1 sec. | 1/ 0 | 1 | 1 sec. | 1 / 0 |
| Rd53 | 54 | 6 | 1 min. | 10.1/ 1 .3 | 6 | 12.30 min. | 11.8/ 2.4 |
| Rd73 | 63 | 63 | 2 sec. | 63 / 0 | 63 | 5 min. | 72.8 / 7.1 |
| Squar5 | 32 | 23 | 1 sec. | 24 /1.1 | 23 | 2 sec. | 24.3/ 1.36 |
| Sym10 | 187 | 64 | 1 min. | 86/ 16.7 | 64 | ~ 22 hours | 91.5/ 28.6 |
| 9sym | 92 | 34 | 40 sec. | 52 /8.9 | 36 | ~ 12 hours | 56.6/18 .2 |
| life | 372 | 40 | 2 sec.. | 44/ 8 .2 | 48 | ~ 16 hours | 62/11.6 |
| clip | 80 | 182 | 8 sec. | 182 / 0 | 188 | ~ 6 hours | 209/ 12.8 |
| Newtag | 22 | 1 | 16 sec. | 2 / 0.5 | 1 | 3 min. | 4 / 1.2 |

## REFERENCES


[1] T. Sasao, " Logic synthesis and optimization ", Kluwar Academic Publishers, Boston/London/Dordrecht, 1993 , ISBN: 0-7923-9308-2.

[2] D.H. Green, "Reed-Muller variable-entered vectors and map", Int. J. Electronics, Vol. 78, No. 1, 1995, pp. 161-186.

[3] D.H. Green, "Reed-Muller canonical forms with mixed polarity and their manipulations", IEE Proceedings, Part E: Computers and Digital Techniques, Vol. 137, No. 1, 1990, pp. 103-113.

[4] L. McKenzie, A.E.A. Almaini, J.F. Miller, P. Thomson, "Optimization of Reed-Muller logic functions", Int. J. Electronics, Vol. 75, No. 3, 1993, pp. 451-466.

[5] M.K. Habib,"A new approach to generate fixed polarity Reed Muller expansions for completely and incompletely specified functions", INT. J. Electronics, Vol. 89, No. 11, 2002, pp. 845-876.

[6] M.K. Habib, "Efficient and fast algorithm to generate minimal Reed-Muller Exclusive-OR expansions with mixed polarity for completely and Incompletely specified function and its computer implementation", Computers Elect. Eng., Vol. 19, No. 3, 1993, pp. 193-211.

[7] M. Helliwell , M. Perkowski, "Fast algorithm to minimize multi output mixed-polarity generalized Reed-Muller forms", Proc 25th IEEE/ACM Conf. on Design Automation, Anaheim, CA, 1988, pp. 427-432.

[8] L. Wang, A.E.A. Almaini, "Exact minimisation of large multiple output FPRM functions", IEE Proc.-Computer Digital Tech., , Vol. 149, No. 5, Sep. 2002.

[9] A.E.A. Almaini, N. Zhuang , "Variable ordering of BDDs for multioutput boolean functions usinge evolutionary techniques". Fourth IEEE-ICECS'97 conf. , Cairo, EGYPT, December 1997, pp. 1239-1244.

[10] Y. Xia, X**.** Ye, L. Wang, Z. Zou, A.E.A. ALmaini, " Novel synthesis and optimization of multi - level mixed polarity Reed-Muller functions", J. Computer Science & Technology, Vol. 20, No. 6, Nov. 2005, pp. 895-900.

[11] P. OH, A.E.A. Almaini, "Decision diagram using 2 variable nodes" , WSEAS transactions on circuits and systems, issue 3 , Vol. 6, March 2007.

[12] B.A. AL JASSANI, N. Urquhart, A.E.A Almaini, "Optimization of MPRM functions using tabular techniques and genetic algorithms", MEDJEC, Vol. 4, No. 4, 2008, pp.115-125

[13] S. Yang, "Logic Synthesis and optimization benchmarks user guide." *Technical Report* 3, Microelectronics Center of North Carolina, 1991.

[14] L. Robert , "Logic synthesis and optimization benchmarks user guide" Technical Report 2, Microelectronics Center of North Carolina, Dec. 1988.