©2010 International Journal of Computer Applications (0975 - 8887)

Volume 1 – No. 17

Simplification of Boolean Algebra through DNA Computing

Sanchita Paul
B.I.T Mesra,
Ranchi

Gadadhar Sahoo
B.I.T Mesra
Ranchi

ABSTRACT

DNA Computing utilizes the properties of DNA for performing the computations. The computations include arithmetic and logical operations such as simplification of Boolean expression to its simplest form. Boolean function can be built from ANDs, ORs, and NOTs using minterm expansion. However, a practicing computer engineer will very rarely be satisfied with a minterm expansion, because as a rule, it requires more gates than necessary. The laws and identities of Boolean algebra will almost always allow us to simplify a minterm expansion. The efficiency of a logic circuit is high when the number of logic gates used to build it is small. However, minterm expression may be often simplified to a simpler Boolean expression, which can be implemented with fewer logic gates.

In this paper we introduced a new DNA computing algorithm for reducing any Boolean expression to its simplest form by using DNA strands. The major benefits of this method are its extraordinary information density, vast parallelism and ease of operation. In addition the most merit of this DNA Algorithm is its automation characteristics, and simple coding steps.

Keywords

DNA computing, Minterms, Simplify, DNA Strands, Boolean expression.

1. INTRODUCTION

DNA computing is new computation paradigms, which proposes the use of molecular biology tools to solve different mathematical problems. It is a form of computing which use DNA, biochemistry and molecular biology, instead of the traditional silicon-based computer technologies. DNA computing is interested in applying computer science methods and models to understand such biological phenomena and gain interest into early molecular evolution and origin of biological information processing. The primary advantage of DNA based computation is the ability to handle millions of operations in parallel. DNA computing is fundamentally similar to parallel computing in that it takes advantage of the many different molecules of DNA to try many different possibilities at once.

DNA computing has two important features, which are Watson-Crick complimentarily and massive parallelism. Using the features, we solve some optimization problems, which usually need exponential time on silicon-based computers, in polynomial steps with DNA molecules.

However, for DNA computing to be applicable on a various range of problems for primitive operations, such as logic or arithmetic operations. A number of procedures have been proposed for the primitive operations with DNA molecules [1], [3], [4], [10], [11], [12], [14].

Boolean algebra:

Boolean algebra is algebra for the manipulation of objects that can take on only two values, typically true and false, although it can be any pair of values. Because computers are built as collections of switches that are either "on" or "off," Boolean algebra is a very natural way to represent digital information. In reality, digital circuits use low and high voltages, but for our level of understanding, 0 and 1 will suffice. It is common to interpret the digital value 0 as false and the digital value 1 as true.

Boolean Expression:

A Boolean expression on the Boolean variables $\{x_1, x_2, ..., x_n\}$ is an expression using those variables and the operations of a Boolean algebra. Every Boolean expression defines a Boolean function. Boolean function can be built from ANDs, ORs, and NOTs using minterm expansion.

Simplification of Boolean algebra:

The laws and identities of Boolean algebra will almost always allow us to simplify a minterm expression. The efficiency of a logic circuit is high when the number of logic gates used to build it is small. However, the sum-of-products (minterm) expression may be often simplified to a simpler Boolean expression, which can be implemented with fewer logic gates.

In this paper we introduced a new DNA algorithm for reducing any Boolean expression to its simplest form by using different combination of single strand DNAs. The rest of the paper is organized as follows: in section 2, we introduce the mathematical process to simplify Boolean expression, in section 3, we introduce the DNA algorithm to simplify the Boolean expression and section 4, we will give the example that how to simplify the Boolean Expression by using DNA computing following by conclusion.

2. SIMPLIFICATION OF BOOLEAN ALGEBRA

Some standardized forms are required for Boolean expression to simplify communication of the expression.
Sum-of-products (SOP): Example:
F(A,B,C,D)= AB+BCD+AD
The minterms in this sum correspond to those combinations of the values for which the function has a value of 1. This Boolean sum is sometimes called a **sum of products expansion** or **disjunctive normal form**.
We know that any Boolean function can be built from ANDs, ORs, and NOTs using minterm expansion. However, a practicing computer engineer will very rarely be satisfied with a minterm expansion, because as a rule, it requires more gates than necessary. The laws and identities of Boolean algebra will almost always allow us to simplify a minterm expansion. For example, the minterm expansion for a Boolean function f of three variables might be represented as follows:

$f = x'y'z' + x'y'z + x'yz' + x'yz + xyz' + xyz$

This would require a circuit with maximum gates: 12 ANDs, 5 ORs and 9 NOTs.
Using the identities of Boolean algebra, this minterm expansion can be simplified considerably:

$f = x'y'z' + x'y'z + x'yz' + x'yz + xyz' + xyz$
$= x'y'(z' + z) + x'y(z' + z) + xy(z' + z)$ distributive law
$= x'y' + x'y + xy$ complementarity & identity
$= x'(y' + y) + xy$ distributive law
$= x' + xy$ complementarity & identity
$= x' + y$ redundancy

So, that big long minterm reduces down to $x' + y$ which can be built with 1 OR and 1NOT.

Therefore, we will look at a very simple technique that usually leads to a significant simplification of minterms. It won't always produce the simplest form, but it's close enough for most engineers considering the difficulty of the alternative method.

3. A NEW DNA ALGORITHM FOR SIMPLIFICATION BOOLEAN ALGEBRA

With the help of massive parallelism of DNA hybridization and the complementary Watson-Crick law, the optimal simplified expression can be found by basic molecular operations.

By means of the basic molecular operations such as merge, separate, denature, detect, etc., are used to simplify the Boolean expression to its simplest form.

For any given Boolean expression of ‘n’ variables in sum of products form, first of all we will see that if all the minterms contains ‘n’ variables or not. If not then we expand the expression with the possible combinations of variables left out in that specific minterm.

Algorithm:

Step 1: We choose 4n groups of oligonucleotides divided into four groups. The oligonucleotides of the first group represented variables x_1, $x_2, \dots\dots\dots x_n$. The oligonucleotides of the second group represented variables $\overline{x_1}$, $\overline{x_2}$ ……………$\overline{x_n}$ (where, x=1 if and only if x=0); the oligonucleotides of third and fourth group represented the complementary strands of the first group (that is $x_1', x_2' \dots\dots x_n'$) and second group (that is $\overline{x_1}'$, $\overline{x_2}'$……………$\overline{x_n}'$) respectively.

Step 2: We generate different 2^n combination of single strands DNA molecules for n variable where the oligonucleotides of third and fourth group are ligated according to the 2^n combinations of ‘n’ variables and placed those single stranded DNA molecules in individual 2^n test tubes.

Step 3: Now, the oligonucleotides of first and second groups are ligated according to minterms of the given expression. And AMPLIFY each minterms 2^n times.

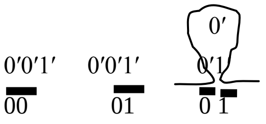
Step 4: Then we MERGE the oligonucleotides of Step 2 and Step 3. The best paired strands are kept and remaining unpaired and semi-paired strands are separated from the test-tubes.

Step 5: The paired strands in each of the test-tube are denatured and keep the oligonucleotide which represent the complementary strands of minterms, in the test tubes and remaining strands are separated from the test tubes.

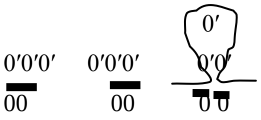
Step 6: Two minterms can be combined if they differ in exactly one literal. This means that their corresponding bit strings differ at exactly one bit position (Example: if 3 literals are present in each minterms then 000 can combined 001, 010, 100 etc.). So, the complementary strands of minterms which we get from step 5 are amplified n times, if one minterm can combined with n minterms, (if they differ in exactly one literal).

Step 7: Now, we have to generate the strands with combination of (n-1) variables and separate those strands in n different test tubes. Because n variables are present in each minterms. (Example: like 001 minterms having 3 combinations that are 00, 01 and 01seperated in three different test tubes).

Step 8: Now we have to combined those two minterms which are differ in exactly one bit position. (Example: like 001 minterms having 3 combinations that are 00, 01 and 01, seperated in three different test tubes. Now in that 3 test tube we merge the complementary strands of 0′0′1′.



Similarly, 001 differ 000 with exactly 1 bit position, so 000 minterms having 3 combination of 2 variables that are 00, 00 and 00, separated in three different test tubes and merge complementary strands of 0′0′ 0′.



Now we have to check that one combination is same for both the cases and that is 00).

Step 9: We store only common terms of two minterms (For above example, that is 00 because for both the cases common ds DNA is 00 and 0'0') and remaining portions are left out from the test tubes.

Step 10: We will denature the ds DNA which we get in step 9 and keep complementary part of the common terms (Example: for above case it is 0′0′) and other are discarded from the test tubes.

Step 11: We store the result.

After getting result we will check any common terms are left between the minterms. If yes then go to next step otherwise go to step 18.

Step 12: Then we have to generate the strands with combination of (n-2) variables and separate those combinations in different test tubes.

Step 13: Now we have to combined those two minterms which are differ in exactly one bit position same as step 8.

Step 14: We store only common terms of two minterms and remaining portions are left out from the test tubes.

Step 15: We will denature the ds DNA and keep complementary part of the common terms in one test tube.

Step 16: We store the result and check that any common terms are left between the minterms which we get from step 15.

Step 17: If yes, then we have to generate the strands with combination of (n-3) variables and go to step 7. Process will continue until no common terms are left.

Step 18: End.

4. Example

For a given three variable expression:

$$Y = \bar{A}\,\bar{B}\,\bar{C} + \bar{A}\bar{B}C + \bar{A}B\bar{C} + \bar{A}BC + AB\bar{C} + ABC$$

Here each minterms having equal number of variables. So there is no need of expansion.

Step 1: We choose 12 oligonucleotides which are divided into 4 groups. The oligonucleotides of the first group represents variables A, B, C; the oligonucleotides of the second group represents $\bar{A}$, B, C; and the third and fourth group represent the complementary strands of first and second group respectively (Denoted as, A′, B′, C′ and $\bar{A}′$, B′, C′).

Step 2: We generate 2^3 combinations of single strands DNA molecules for 3 variables. Where the oligonucleotides of third and fourth group are ligated according to the combinations of '3' variables and placed in individual test tubes.

$$\bar{A}'\bar{B}'\bar{C}' \quad \bar{A}'\bar{B}'C' \quad \bar{A}'B'\bar{C}' \quad \bar{A}'B'C'$$

$$A'\bar{B}'\bar{C}' \quad A'\bar{B}'C' \quad A'B'\bar{C}' \quad A'B'C'$$

Figure 1. 2^3 combination of ss DNA molecules are placed different test tubes

Step 3: Now, the oligonucleotides of first and second group are ligated according to minterms of the given expression. AMPLIFY 8 times for each minterms. Resulting DNA strands for minterms are as follows:

$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$	$\bar{A}B\bar{C}$
$\bar{A}BC$	$A\bar{B}C$	ABC

Figure 2. ss DNA molecules representing minterms of a given Boolean Expression

Step 4: We MERGE the 8 sets of oligonucleotides which represent the minterms of Step 3 and with individual combinations which are already kept in 8 individual test tubes of Step 2. The best paired strands are kept and remaining unpaired and semi-paired strands are discarded from the test-tube.

$\bar{A}\bar{B}\bar{C}$ $\bar{A}'\bar{B}'\bar{C}'$ T_0	$\bar{A}\bar{B}C$ $\bar{A}'\bar{B}'C'$ T_1	$\bar{A}B\bar{C}$ $\bar{A}'\bar{B}'C'$ T_2	$\bar{A}BC$ $\bar{A}'B'C'$ T_3
$A\bar{B}\bar{C}$ T_4	$A\bar{B}C$ T_5	$AB\bar{C}$ $A'B'\bar{C}'$ T_6	ABC $A'B'C'$ T_7

Figure 3. Merge each minterms with different combinations of oligonucleotides

$\bar{A}\bar{B}\bar{C}$ $\bar{A}'\bar{B}'\bar{C}'$ T_0	$\bar{A}\bar{B}C$ $\bar{A}'\bar{B}'C'$ T_1	$\bar{A}B\bar{C}$ $\bar{A}'\bar{B}'C'$ T_2	$\bar{A}BC$ $\bar{A}'B'C'$ T_3
		$AB\bar{C}$ $A'B'\bar{C}'$ T_6	ABC $A'B'C'$ T_7

Figure 4. Unpaired and semi-paired strands are discarded

Step 5: The paired strands are denatured and the oligonucleotides which represent the complementary strands of each minterm are kept in the test tubes. Remaining strands are separated from the test tubes.

$\bar{A}'\bar{B}'\bar{C}'$ T_0	$A'\bar{B}'C'$ T_1	$\bar{A}'\bar{B}'C'$ T_2	$\bar{A}'B'C'$ T_3
		$A'B'\bar{C}'$ T_6	$A'B'C'$ T_7

Figure 5. The paired strands are denatured and the complementary strands of each minterm are kept in test tubes

Step 6: Two minterms can be combined if they differ in exactly one literal. This means that their corresponding bit strings differ at exactly one bit position. For the above example, we can compare T_0 - T_1 and T_0-T_2, T_1 - T_3 , T_2 - T_3 and T_2 – T_6 and finally T_3 - T_7 and T_6 - T_7 because in those test tubes the strands having bit string difference is exactly by 1 bit position.

The complementary strands of minterms which we get from step 5 are amplified.

AMPLIFY (T_0, T_0', T_0'', T_{00}, T_{00}', T_{00}'' , T_{000}, T_{000}', T_{000}'')

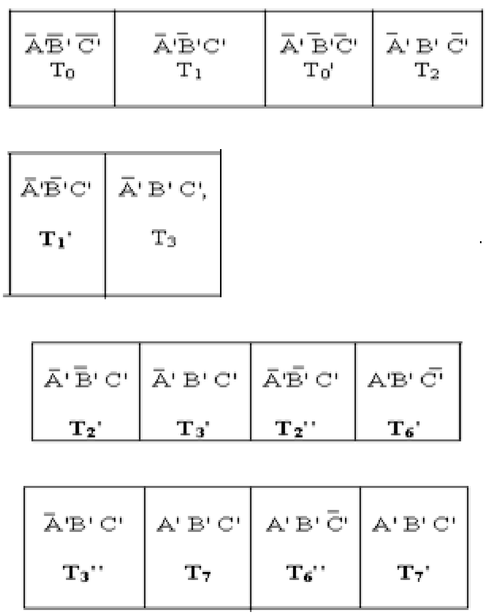
AMPL1IFY (T_1, T_1', T_1'', T_{10}, T_{10}', T_{10}'', T_{100}, T_{100}', T_{100}'')

AMPLIFY (T_2, T_2', T_2'', T_{20}, T_{20}', T_{20}'', T_{200}, T_{200}', T_{200}'')

AMPLIFY (T_3, T_3', T_3'', T_{30}, T_{30}', T_{30}'', T_{300}, T_{300}', T_{300}'')

AMPLIFY (T_6, T_6', T_6'', T_{60}, T_{60}', T_{60}'', T_{600}, T_{600}', T_{600}'')

AMPLIFY (T_7, T_7', T_7'', T_{70}, T_{70}', T_{70}'', T_{700}, T_{700}', T_{700}'')



$\overline{A}'\overline{B}'\overline{C}'$ T_0	$\overline{A}'\overline{B}'C'$ T_1	$\overline{A}'\overline{B}'\overline{C}'$ T_0'	$\overline{A}'B'\overline{C}'$ T_2

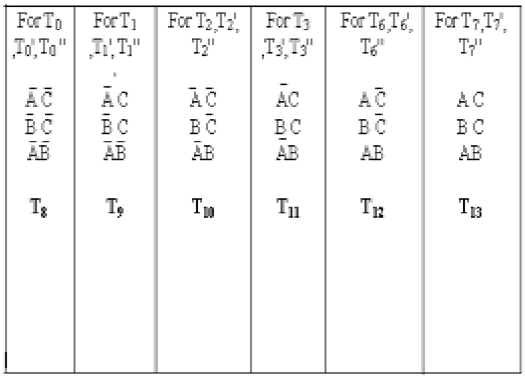
$\overline{A}'\overline{B}'C'$ T_1'	$\overline{A}'B'C'$, T_3

$\overline{A}'\overline{B}'C'$ T_2'	$\overline{A}'B'C'$ T_3'	$\overline{A}'\overline{B}'C'$ T_2''	$A'B'\overline{C}'$ T_6'

$\overline{A}'B'C'$ T_3''	$A'B'C'$ T_7	$A'B'\overline{C}'$ T_6''	$A'B'C'$ T_7'

Figure6: Combined two minterms if they differ in exactly one literal

Step 7: Now, generate the strands with combination of 2 variables and separate those strands in three different test tubes and amplify those test tubes according to the minterms. For test tubes T_0, T_0' and T_0'' where, $\overline{A}$=0, $\overline{B}$=0, $\overline{C}$=0, from that, we will take the combination of two variables and those are $\overline{AB}$, $\overline{BC}$, $\overline{AC}$ and put it into 1 test tube.



For T_0, T_0', T_0''	For T_1, T_1', T_1''	For T_2, T_2', T_2''	For T_3, T_3', T_3''	For T_6, T_6', T_6''	For T_7, T_7', T_7''
$\overline{A}\,\overline{C}$	$\overline{A}\,C$	$\overline{A}\,\overline{C}$	$\overline{A}C$	$A\,\overline{C}$	$A\,C$
$\overline{B}\,\overline{C}$	$\overline{B}\,C$	$B\,\overline{C}$	$B\,C$	$B\,\overline{C}$	$B\,C$
$\overline{A}\overline{B}$	$\overline{A}\overline{B}$	$\overline{A}B$	$\overline{A}B$	AB	AB
T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}

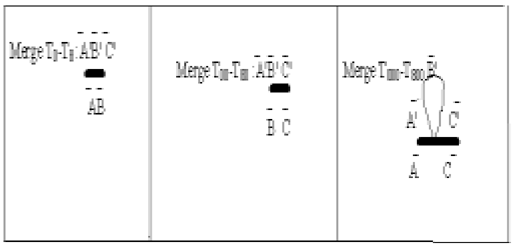
Figure7: Combination of 2 variables

SEPARATE (T_8, T_{80}, T_{800})
AMPLIFY (T_8, T_{80}, T_{800}, T_8', T_{80}', T_{800}', T_8'', T_{80}'', T_{800}'') [T_8: $\overline{AB}$, T_{80}: BC, T_{800}: $\overline{AC}$]
SEPARATE (T_9, T_{90}, T_{900})
AMPLIFY (T_9, T_{90}, T_{900}, T_9', T_{90}', T_{900}', T_9'', T_{90}'', T_{900}'') [T_9: B C, T_{90}: $\overline{AB}$, T_{900}: $\overline{A}C$]
SEPARATE (T_{10}, T_{100}, T_{1000})
AMPLIFY (T_{10}, T_{100}, T_{1000}, T_{10}', T_{100}', T_{1000}', T_{10}'', T_{100}'', T_{1000}'')
SEPARATE (T_{11}, T_{110}, T_{1100})
AMPLIFY (T_{11}, T_{110}, T_{1100}, T_{11}', T_{110}', T_{1100}', T_{11}'', T_{110}'', T_{1100}'')
SEPARATE (T_{12}, T_{120}, T_{1200})
AMPLIFY (T_{12}, T_{120}, T_{1200}, T_{12}', T_{120}', T_{1200}', T_{12}'', T_{120}'', T_{1200}'')
SEPARATE (T_{13}, T_{130}, T_{1300})
AMPLIFY (T_{13}, T_{130}, T_{1300}, T_3', T_{130}', T_{1300}', T_{13}'', T_{130}'', T_{1300}'')

Step 8: Now, we have to combine those two minterms which are differ in exactly one bit position.

Compare T_0 - T_1

MERGE [T_0-T_8, T_{00}-T_{80}, T_{000}-T_{800} (For T_0 test tube)



MERGE [T_1-T_9, T_{10}-T_{90}, T_{100}-T_{900} (For T_1 test tube)]

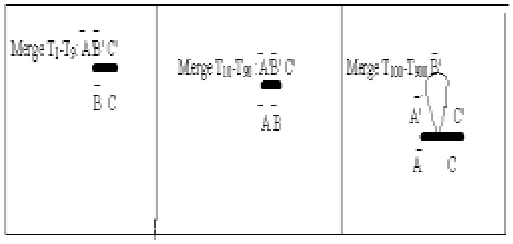


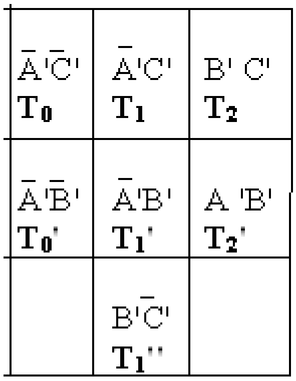
Figure8: Compare T_0 - T_1 and get result AB

Compare T_0 and T_1 test tubes, both the cases $\overline{\mathbf{A}}\overline{\mathbf{B}}$ is common.

<u>Compare T_0' and T_2:</u>
MERRGE [T_0'-T_8', T_{00}'-T_{80}', T_{000}'-T_{800}' (For T_0' test tube)]
MERGE [T_2-T_{10}, T_{20}-T_{100}, T_{200}-T_{1000} (For T_2 test tube)]
Compare T_0' and T_2 test tubes, both the cases $\overline{\mathbf{A}}\overline{\mathbf{C}}$ is common.
<u>Compare T_1' and T_3:</u>
MERRGE [T_1'-T_9', T_{10}'-T_{90}', T_{100}'-T_{900}' (For T_1' test tube)]
MERGE [T_3-T_{11}, T_{30}-T_{110}, T_{300}-T_{1100} (For T_3 test tube)
Compare T_0' and T_2 test tubes, both the cases $\overline{\mathbf{A}}\mathbf{C}$ is common.
<u>Compare T_2' and T_3':</u>
MERRGE [T_2'-T_{10}', T_{20}'-T_{100}', T_{200}'-T_{1000}' (For T_2' test tube)]
MERGE [T_3'-T_{11}', T_{30}'-T_{110}', T_{300}'-T_{1100}' (For T_3' test tube)]
Compare T_2' and T_3' test tubes, both the cases $\overline{\mathbf{A}}\mathbf{B}$ is common.
<u>Compare T_2'' and T_6':</u>
MERRGE [T_2''-T_{10}'', T_{20}''-T_{100}'', T_{200}''-T_{1000}'' (For T_2'' test tube)]
MERGE [T_6'-T_{12}', T_{60}'-T_{120}', T_{600}'-T_{1200}' (For T_6' test tube)]
Compare T_2'' and T_6' test tubes, both the cases **BC** is common.
<u>Compare T_3'' and T_7:</u>
MERRGE [T_3''-T_{11}'', T_{30}''-T_{110}'', T_{300}''-T_{1100}'' (For T_3'' test tube)]
MERGE [T_7-T_{13}, T_{70}-T_{130}, T_{700}-T_{1300} (For T_7 test tube)]
Compare T_3'' and T_7 test tubes, both the cases **BC** is common.
<u>Compare T_6'' and T_7':</u>
MERRGE [T_6''-T_{12}'', T_{60}''-T_{120}'', T_{600}''-T_{1200}'' (For T_6'' test tube)]
MERGE [T_7'-T_{13}', T_{70}'-T_{130}', T_{700}'-T_{1300}' (For T_7' test tube)]
Compare T_3'' and T_7 test tubes, both the cases **AB** is common.

<u>Step 9:</u> We store only common terms of two minterms others are separated from the test tubes. In common terms, we store only ds DNA part and remaining portions are left out by cutting through restriction enzymes.
<u>Step 10:</u> We will denature the ds DNA and keep complementary part of the common terms in a test tubes.
And the common terms are $A'B'+B'C'+B'C'+A'C'+A'C'+A'B'+A'B'$
<u>Step 11:</u> We store the result and check if any common terms are left between the minterms, which we get from step 10. If yes then go to next step otherwise go to step 18.

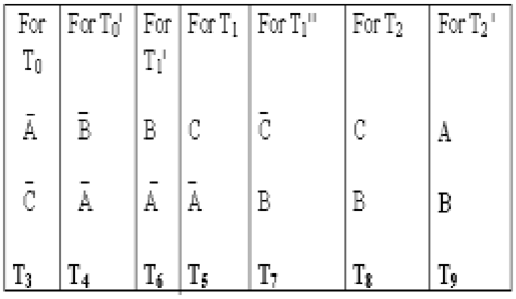
After denaturation, we get those minterms terms having common terms, so again the same process will be continued. Store the minterms in different test tubes.



$\bar{A}'\bar{C}'$ T_0	$\bar{A}'C'$ T_1	$B'\ C'$ T_2
$\bar{A}'\bar{B}'$ T_0'	$\bar{A}'B'$ T_1'	$A\ 'B'$ T_2'
	$B'\bar{C}'$ T_1''	

Figure9: Minterms that consist some common terms after first iteration

<u>Step 12:</u> Then, we generate the strands with combination of 1 variable and separate those combinations in different test tubes.



For T_0	For T_0'	For T_1'	For T_1	For T_1''	For T_2	For T_2'
$\bar{A}$	$\bar{B}$	B	C	$\bar{C}$	C	A
$\bar{C}$	$\bar{A}$	$\bar{A}$	$\bar{A}$	B	B	B
T_3	T_4	T_6	T_5	T_7	T_8	T_9

Figure10: Combination of 1 variable

<u>Step 13:</u> We have to combine those two minterms which are differ in exactly one bit position. For the above example, we can compare, T_0 - T_1, T_0-T_1', T_0-T_1'' and T_0' - T_1, T_0'- T_1' and T_1'- T_2' , T_1''-T_2, $T_1 – T_2$.
The complementary strands of minterms which we get from step 10 are amplified.

AMPLIFY (T_0, T_{01}, T_{02}, T_{03}, T_{04}, T_{05}, T_{06}, T_{07}, T_{08}.)
AMPLIFY (T_0', T_{01}', T_{02}', T_{03}', T_{04}', T_{05}', T_{06}', T_{07}', T_{08}')
AMPLIFY (T_1, T_{11}, T_{12}, T_{13}, T_{14}, T_{15}, T_{16}, T_{17}, T_{18})
AMPLIFY (T_1', T_{11}', T_{12}', T_{13}', T_{14}', T_{15}', T_{16}', T_{17}', T_{18}')
AMPLIFY (T_1'', T_{11}'', T_{12}'', T_{13}'', T_{14}''., T_{15}'', T_{16}'', T_{17}'', T_{18}'')
AMPLIFY (T_2, T_{21}, T_{22}, T_{23}, T_{24}, T_{25}, T_{26}, T_{27}, T_{28})
AMPLIFY (T_2', T_{22}', T_{23}', T_{24}', T_{25}', T_{26}, ′ T_{27}, T_{28}')

<u>Step 14:</u> Now, we have to generate the strands with combination of 1 variable and separate those strands in 2 different test tubes and amplify those test tubes according to the minterms.
SEPARATE (T_3, T_{30})
AMPLIFY (T_3, T_{30}, T_3', T_{30}', T_3'', T_{30}'')
SEPARATE (T_4, T_{40})
AMPLIFY (T_4, T_{40}, T_4', T_{40}', T_4'', T_{40}'')
SEPARATE (T_5, T_{50})
AMPLIFY (T_5, T_{50}, T_5', T_{50}', T_5'', T_{50}'')
SEPARATE (T_6, T_{60})
AMPLIFY (T_6, T_{60}, T_6', T_{60}', T_6'', T_{60}'')
SEPARATE (T_7, T_{70})
AMPLIFY (T_7, T_{70}, T_7', T_{70}', T_7'', T_{70}'')
SEPARATE (T_8, T_{80})
AMPLIFY (T_8, T_{80}, T_8', T_{80}', T_8'', T_{80}'')
SEPARATE (T_9, T_{90})
AMPLIFY (T_9, T_{90}, T_9', T_{90}', T_9'', T_{90}'')
Now, those two minterms which differ in exactly one bit position are combined.

<u>Compare T_0 and T_1:</u>
MERRGE [T_0-T_3, T_{01}-T_{30}]
MERGE [T_1-T_5, T_{11}-T_{50}]
Compare T_0 and T_1 test tubes, both the cases $\bar{\mathbf{A}}$ is common

<u>Compare T_0 and T_1':</u>
MERRGE [T_{02}-T_3', T_{03}-T_{30}']
MERGE [T_1'-T_6, T_{11}'-T_{60}]
Compare T_0 and T_1' test tubes, both the cases $\bar{\mathbf{A}}$ is common

<u>Compare T_0' - T_1:</u>
MERRGE [T_0'-T_4, T_{01}'-T_{40}]
MERGE [T_{12}- T_5', T_{13}- T_{50}']
Compare T_0' and T_1 test tubes, both the cases $\bar{\mathbf{A}}$ is common

<u>Compare T_0' - T_1':</u>
MERRGE [T_{02}'-T_4', T_{03}'-T_{40}']
MERGE [T_{12}'- T_6, T_{13}'- T_{60}]
Compare T_0' and T_1' test tubes, both the cases $\bar{\mathbf{A}}$ is common

Compare T_1'' - T_2:
MERRGE [T_1''-T_7, T_{11}''-T_{70}]
MERGE [T_2- T_8, T_{21}- T_{80}]
Compare $T_1''$and T_2 test tubes, both the cases **B** is common
Compare T_0- T_1'':
MERRGE [T_{04}-T_3'', T_{05}- T_{30}'']
MERGE [T_{12}''- T_7', T_{13}''- T_{70}']
Compare $T_1''$and T_0 test tubes, both the cases $\overline{\mathbf{C}}$ is common
Compare T_2'- T_1':
MERRGE [T_{14}'-T_6', T_{15}'- T_{60}']
MERGE [T_2'- T_9, T_{21}'- T_{90}]
Compare $T_2'$and T_1' test tubes, both the cases is **B** common
Compare T_1- T_2:
MERRGE [T_{15}-T_5'', T_{15}- T_{50}'']
MERGE [T_{22}- T_8, T_{23}- T_{80}]
Compare T_1 and T_2 test tubes, both the cases **C** is common
Result is: $\overline{A}+\overline{A}+\overline{A}+\overline{A}+B+B+C+\overline{C}$
After denaturation, we get those minterms terms having common terms, so again the same process will be continued. Store the minterms in different test tubes and repeat the steps as above. The final result will be $\overline{A}$+B

5. CONCLUSION

The purpose of this chapter was to show a mathematical application of DNA computing. Hence, we consider DNA as Arithmetic-Logic Unit, where human operators implement bio-chemistry procedures to perform mathematical operations. The power of the DNA Computing consists in the capability to represent, and compute, huge binary numbers, or highly small ones which are impossible to consider in a conventional computer. In other words, we are able to calculate mathematical operations with unlimited decimal digits. It is worthy noticing that "unlimited" does not mean "endless", but unfixed number of bits. If we consider that 50g of DNA contains 10^{33} molecules, it is clear that in few grams of DNA we can encode a great deal of molecular bits. In this paper we introduced a new DNA computing algorithm for reducing any Boolean expression to its simplest form by using DNA strands. This feature is a beautiful remedy for computational problems, which all depend on the fixed number of bits reserved to the representation in conventional computing.

6. REFERENCES

[1] Fukagaw. H., Fujiwara. A., Procedures for multiplication and division in DNA computing, 2004.

[2] Adleman L. M., Computing with DNA. Scientific American, 279(2):54–61, 1998.

[3] Gupta. V., Parthasarathy. S., and Zaki. M. J., Arithmetic and logic operations with DNA. In Proceedings 3rd DIMACS Workshop on DNA Based Computers, pages 212–220, 1997.

[4] Hug. H., and Schuler. R., DNA-based parallel computation of simple arithmetic. In Proceedings of International Meeting on DNA Based Computers, pages 159–166, 2001.

[5] Liption. R. J., DNA solution of hard computational problems. Science, 268:542–545, 1995.

[6] Qiu. Z. F., and Lu. M., Arithmetic and logic operations for DNA computers. In Proceedings of the Second IASTED International conference on Parallel and Distributed Computing and Networks, pages 481–486, 1998.

[7] Qiu. Z. F., and Lu. M., Take advantage of the computing power of DNA computers. In Proceedings of the Third Workshop on Bio-Inspired Solutions to Parallel Processing Problems, IPDPS 2000 Workshops, pages 570–577, 2000.

[8] Reif. J. H., Parallel bimolecular computation: Models and simulations. Algorithmica, 25(2/3): 142–175, 1999.

[9] Merrifield. R. B., Solid phase peptide synthesis. I. The synthesis of a tetra peptide. Journal of the American Chemical Society, 85:2149–2154, 1963.

[10] Frisco. P., Parallel arithmetic with splicing. Romanian Journal of Information Science and Technology, 2(3):113–128, 2000.

[11] Fujiwara. A., Matsumoto. K., and Chen. W., Addressable Procedures for logic and arithmetic operations with DNA molecules. International Journal of Foundations of Computer Science, 15(3):461–474, 2004.

[12] Guarnieri, F., Fliss. M., and Bancroft. C., Making DNA add. Science, 273:220–223, 1996.

[13] Pˇaun. G., Rozeberg. G., and Salomaa. A., DNA computing. Springer-Verlag, 1998.

[14] Kamio. S., Takehara A., and Fujiwara. A., Procedures for computing the maximum with dna strands. In Proceedings of the 2003 International Conference on Parallel and Distributed Processing Techniques and Applications, volume 1, pages 351–357, 2003.