A Dissertation

On

Evolutionary Logic Circuit Synthesis Based On Mutual Information Based Fitness Function

Submitted in Partial fulfillment of requirement for the Award of Degree of

MASTER OF ENGINEERING

(Electronics and Communication)

Submitted By

ArunRudra

College Roll No. 01/E&C/PT/2009

University Roll No. 13921

Under the guidance of

Dr(Ms) S Indu

Associate Professor

&

Dr (Ms) Neeta Pandey

Associate Professor

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

DELHI COLLEGE OF ENGINEERING

DELHI UNIVERSITY

2009-2012

Certificate

Certified that the thesis work entitle “**Evolutionary Logic Circuit Synthesis Based On Mutual Information Based Fitness Function”** is bonafide work carried out by **Mr. ArunRudra** in partial fulfillment of Master’s Degree in Engineering in Electronics and Communication to University of Delhi during the year 2009-2012. The project report has been approved as it satisfies the academic requirements in respect of thesis work prescribed for the Masters of Engineering Degree.

Signature of Guide:-	Signature of Guide:-
Dr(Ms) S Indu	**Dr (Ms) Neeta Pandey**
Associate Professor	Associate Professor
Dept of Electronics& Comm.	Dept of Electronics & Comm.
Delhi College of Engineering	Delhi College of Engineering

Acknowledgement

It is a great pleasure to have the opportunity to extend my heartfelt gratitude to everybody who helped me throughout the course of this project.

It is distinct pleasure to express my deep sense of gratitude and in indebtness to my learned supervisor **Dr. S Indu, Associate Professor andDr. Neeta Pandey, Associate Professor at DCE**for their invaluable guidance, encouragement and patient reviews. Without their guidance and reviews it would have been very difficult for me to complete my project successfully. I am very thankful to **Prof. Rajiv Kapoor, HOD, Dept. of Electronics & Communication at DCE**, who allowed me to do the project under the guidance of **Dr.(Ms) S Indu and Dr (Ms) Neeta Pandey**.

ArunRudra

M.E (Electronics & Communication)
College Roll No. 01/E&C/PT/2009
University Roll No. 13921
Dept. of Electronics & Communication
Delhi College of Engineering

ABSTRACT

Evolutionary Algorithms (EAs) cover all the applications involving the use of Evolutionary Computationin electronic system design. It is largely applied to complex optimizationproblems. EAs introduce a new idea for automatic design of electronic systems; instead of imaginemodel, abstractions, and conventional techniques, it uses search algorithm to design a circuit.

In this project we have used Genetic Algorithm (GA) as the Evolutionary Algorithm and Entropybased measures, such as Mutual Informationand Normalized MutualInformation are investigated as toolsfor similarity, measures between the target and evolving circuit.

The target circuit evolved is combinational logic circuit, a Majority Function using Genetic algorithm and Information Theory Measures.The circuit evolved uses only 2x1 multiplexers and the evolved circuit is compared to the circuit identified using the ROBDD (Reduced Ordered Binary Decision Diagram)

Table of Contents

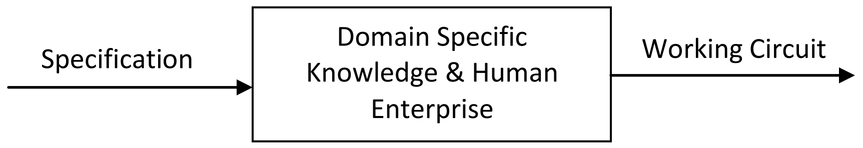
I. Introduction

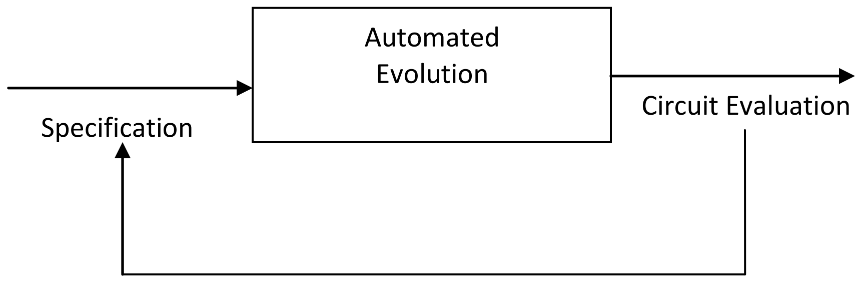
Evolving Algorithms are capable of evolving 100% functional arithmetic circuits. Evolutionary Algorithms include Genetic Algorithms (GA), Genetic Programming (GP), Evolutionary Strategies etc.In our study we will be concentrating on Genetic Algorithm. The largest of these circuits arethe most complex digital circuits to have been designed by purely evolutionary means. The algorithm isable to re-discover conventionally optimum designs for the combinational circuits like full adder and parity generators, but moresignificantly is able to improve on the conventional designs. By analyzing thehistory of an evolving design up to complete functionality it is possible to gain insight into evolutionaryprocess.

The design of electronic circuits is generally a complex task requiring knowledge of large collectionsof domain-specific rules. In particular the process of implementing a digital electronic circuit inhardware has typically involved transforming the original logical specification into a form suitable for, thetarget technology (i.e. choosing the gate types), minimising the representation, optimising therepresentation with respect to user defined constraints (i.e. timing characteristics, fan-in/outs, etc.) andfinally carrying out technology mapping onto the target device. This latter step typically involvesplacing and routing of the component gates which comprise the complete design. It should beemphasised that during all these stages great care has to be taken to maintain the logical functionality ofthe original circuit specification.This new approach is perhaps best expressed as a black-box view of the problem. In this view oneregards the problem of implementing the circuit as being equivalent to designing a black-box withinputs and outputs with the property that on presentation of the original input signals the desired outputsare delivered. The key new feature of this technique is that the details inside the box are encoded into chromosomes and subjected to the usual processes of evolutionary algorithms. In this technique thefitness of a particular chromosome is measured purely as the degree to which the black-box outputsbehave in the desired way.

Up until now, most electronic systems of any complexity were created by a designer who had beentrained in a particular way to understand the operation of individual electronic components, and whowould, therefore, be able to use these rules of behavior to construct larger systems from the basicparts. This was true whether the system to be created was purely analogue (responding to real-worldsignals), purely digital (responding to binary streams), or some combination of the two.

This method of working is somewhat constrained both by the training and experience of the designerand by the domain-specific knowledge which he may or may not possess. For example, some designerswill be more expert in the analogue domain, some more expert in the digital domain. Instead, those whoadvocate the use of evolution to assist in the design process are not so concerned with this type ofexpertise, but merely seek to set up the appropriate conditions which will allow solution to evolvenaturally. Figure demonstrates the difference between the two approaches.





The reason for the recent increase of research activity in the evolvable hardware field is probably due in availability of programmable electronic components. Unlike traditionalcomponents, these devices have no fixed operation or functionality when first obtained. Instead it is theresponsibility of the user/designer to decide - via the appropriate programming - what that functionalityshould be either during or after implementation within a given system. In many instances, these devices,once programmed, are even then not dedicated to that particular operational characteristic, but mayafterwards be re-programmed to adopt yet another different functionality.

We intend to evolve a 3,2 majority function combinational circuit using Genetic Algorithm as the Evolutionary Algorithm. The circuit is evolved using a programmable module written in Verilog HDL called the Universal Module for 3 variables, it can implement any combinational circuit of 3 variables depending upon the fact that how it is programmed. The fitness function used in the evolution is based upon the Information Theory indices of Normalised Mutual Information.
The Evolution process is carried out in real time using MATLAB EDA simulator link and Modelsim.

II. Related Works

I. Mutual Information-based Fitness Functions forEvolutionary Circuit Synthesis[7]

Entropy-based measures, such as Mutual Information and Normalized Mutual Information are investigated as tools for similarity, measures between the target and evolving circuit. Three fitness functions are built over a primitive one. It is shown that the search landscape of Normalized Mutual Informationlooks more amenable for evolutionary computation algorithmsthan simple Mutual Information. The evolutionary synthesized circuits are compared to the known optimum size. A discussion of the potential of the InformationTheoretical approach is given in this paper.

II. Evolutionary Synthesis of Logic Functions using Multiplexers[8]

This paper presents a genetic programming based approach to the synthesis of logic functions by means of multiplexers. This method uses 1-control line multiplexer as the only design unit for the synthesis of any logic function. Logic design with multiplexers is similar to logic design with binary decision diagrams which can be transformed into ordered binary decision diagrams. It is argued that since the metric of the designs is minimum number of components, ordered diagrams are not suitable approach for this particular goal.

III. Gate level synthesis of Boolean Functions using Binary Multiplexers and Genetic Programming [9]

In this paper genetic programming approach for synthesis of logic functions by means of multiplexers is presented. This approach uses 1-control line multiplexers as the only deign unit. Any logic function defined by the truth table can be produced through the replication of this single unit. It's fitness function works in two stages, first it finds the feasible solution and then it concentrates on minimization of the circuit. The proposed approach does not require any knowledge of the application domain.

IV. Information Theory Method forFlexible Network Synthesis [10]

This paper introduces a novel approach to extend flexibility of combinational multi-level networks synthesis based on InformationTheoretical Measure (ITM). This problem is related to optimization for combinational multi-level networks, artificial evolution and machine learning in circuitry design. Using ITMs, we

verify not only that an evolved network achieves the target functionality, but also that this network can. be corrected in a simple regular way to achieve it.
We demonstrate experimental results by evolutionary strategy on gate-level network design: effectiveness in evolved valid networks increases dozens of times.

V. Evolutionary Algorithms and Their Use in the Design of Sequential Logic Circuits[11]

In this paper an approach based on an evolutionary algorithm to design synchronoussequential logic circuits with minimum number of logic gates is suggested. The proposed method consists offour main stages. The first stage is concerned with the use of genetic algorithms (GA) for the state assignmentproblem to compute optimal binary codes for each symbolic state and construct the state transitiontable of finite state machine (FSM). The second stage defines the subcircuits required to achieve the desiredfunctionality. The third stage evaluates the subcircuits using extrinsic Evolvable Hardware (EHW).During the fourth stage, the final circuit is assembled. The obtained results compare favorably againstthose produced by manual methods and other methods based on heuristic techniques.

VI. Application of Design Style in EvolutionaryMulti-Level Networks Synthesis[12]

This paper considers evolutionary design oflogical networks from the Computer Aided Design(CAD) point of view. It states that scanning of a spaceof all possible network solutions by a scanning windowis the crucial point of an evolutionary paradigm.This is the base for implementation of CAD methodsin order to improve the recently obtained results onevolutionary approach for a network synthesis. Firstly, it introduces the concept of a target design stylein evolutionary network synthesis and show that itis closely related to the CAD problem of multi-levelnetworks design over a fixed library of cells. Secondly,because the network search space is partition able,we use the technique **of** decomposition **of** switchingfunctions. Therefore, independent parallel processingof subspaces via genetic algorithms (GAs) is possible.Moreover, since GA is inherently parallel, itachievemassive parallel processing. The experimental datademonstrate the efficiency of the proposed approach andlarge improvements over recently obtained results.

VII. Binary Decision Diagrams [13]

This paper describes a method for defining, analyzing,testing, and implementing large digital functions by means of abinary decision diagram. This diagram provides a complete, concise,"implementation-free" description of the digital functions involved.Techniques are then outlined for using the diagrams to analyze thefunctions involved, for test generation, and for obtaining variousimplementations. It is shown that the diagrams are especially suitedfor processing by a computer. Finally, methods are described forintroducing inversion and for directly "interconnecting" diagrams todefine still larger functions.

VIII. Graph-Based Algorithms for Boolean FunctionManipulation[14]

In this paper a new data structure forrepresenting Boolean functions and an associated set of manipulation algorithms. Functions are represented by directed, acyclicgraphs in a manner similar to the representation, but with further restrictions on theordering of decision variables in the graph. Although a function
requires, in the worst case, a graph of size exponential in thenumber of arguments, many of the functions encountered intypical applications have a more reasonable

representation. Thesealgorithms have time complexity proportional to the sizes of thegraphs being operated on, and hence are quite efficient as long asthe graphs do not grow too large. The experimental resultsfrom applying these algorithms to problems in logic designverification that demonstrates the practicality of this approach.

III. Genetic Algorithm

1. Introduction

Genetic Algorithms are a family of computational models inspired by evolution. These algorithms encodea potential solution to a specific problem on a simple chromosome-like data structure and applyrecombination operators to these structures as to preserve critical information. Genetic algorithmsare often viewed as function optimizer, although the range of problems to which genetic algorithms havebeen applied are quite broad.An implementation of genetic algorithm begins with a population of (typically random) chromosomes.

One then evaluates these structures and allocated reproductive opportunities in such a way that thesechromosomes which represent a better solution to the target problem are given more chances to `reproduce'than those chromosomes which are poorer solutions. The 'goodness' of a solution is typically defined withrespect to the current population.

1.1.Background

Many human inventions were inspired by nature. Artificial neural networks is one example,anotherexample is Genetic Algorithms (GA). GAs search by simulating evolution, starting from an initial set ofsolutions or hypotheses, and generating successive "generations" of solutions. This particular branch ofAI was inspired by the way living things evolved into more successful organisms in nature. The main idea is survival of the test, a.k.a. natural selection.A chromosome is a long, complicated thread of DNA (deoxyribonucleic acid). Hereditary factors that determine particular traits of an individual are strung along the length of these chromosomes, like beadson a necklace. Each trait is coded by some combination of DNA (there are four bases, A (Adenine), C(Cytosine), T (Thymine) and G (Guanine). Like an alphabet in a language, meaningful combinations ofthe bases produce special instructions to the cell.Changes occur during reproduction. The chromosomes from the parents exchange randomly by aprocess called crossover. Therefore, the offspring exhibit some traits of the father and some traits of themother. A rarer process called mutation also changes some traits. Sometimes an error may occur duringcopying of chromosomes (mitosis). The parent cell may have -A-C-G-C-T- but an accident may occurand changes the new cell to -A-C-T-C-T-. Much like a typist copying a book, sometimes a few mistakesare made. Usually this results in a nonsensical word and the cell does not survive. But over millions ofyears, sometimes the accidental mistake produces a more beautiful phrase for the book, thus producinga better species.

1.2.Natural Selection.

In nature, the individual that has better survival traits will survive for a longer period of time. This inturn provides it a better chance to produce offspring with its genetic material. Therefore, after a longperiod of time, the entire population will consist of lots of genes from the superior individuals and lessfrom the inferior individuals. In a sense, the fittest survived and the unfit died out. This force of natureis called natural selection.The existence of competition among individuals of a species was recognized certainly before Darwin.The mistake made by the older theorists (like Lamarck) was that the environment had an effect on anindividual. That is, the environment will force an individual to adapt to it. The molecular explanationof evolution proves that this is biologically impossible. The species does not adapt to the environment,rather, only the fittest survive.

1.3.Simulated Evolution

To simulate the process of natural selection in a computer, we need to define the following: A representationof an individual,at each point during the search process we maintain a "generation" of "individuals". Each individual is a data structure representing the "genetic structure" of a possible solution or hypothesis.

Like a chromosome, the genetic structure of an individual is described using a fixed, finite alphabet.In GAs, the alphabet 0, 1 is usually used. This string is interpreted as a solution to the problem we aretrying to solve.For example, say we want to find the optimal quantity of the three major ingredients in a recipe (say, sugar, wine, and sesame oil). We can use the alphabet 1, 2, 3 ..., 9 denoting the number of ounces ofeach ingredient. Some possible solutions are 1-1-1, 2-1-4, and 3-3-1.As another example, the traveling salesperson problem is the problem of finding the optimal path totraverse, say, 10 cities. The salesperson may start in any city. A solution is a permutation of the 10cities: 1-4-2-3-6-7-9-8-5-10.

As another example, say we want to represent a rule-based system. Given a rule such as "If color=redand size=small and shape=round then object=apple" we can describe it as a bit string by first assumingeach of the attributes can take on a fixed set of possible values. Say color=red, green, blue, size=small,big, shape=square, round, and fruit=orange, apple, banana, pear. Then we could represent the value for each attribute as a sub-string of length equal to the number of possible values of that attribute. Forexample, color=red could be represented by 100, color=green by 010, and color=blue by 001. Note alsothat we can represent color=red or blue by 101, and any color (i.e., a "don't care") by 111. Doing thisfor each attribute, the above rule might then look like: 100 10 01 0100. A set of rules is then represented by concatenating together each rule's 11-bit string. For another example see page 620 in the textbookfor a bit-string representation of a logical conjunction.

1.4.Genetic algorithm vocabulary

Explanation of Genetic Algorithm terms:

Genetic Algorithms	**Explanation**
Chromosome(string, individual)	Solution (coding)
Genes (bits)	Part of solution
Locus	Position of gene
Alleles	Values of gene
Phenotype	Decoded solution
Genotype	Encoded solution

2. Fitness Function

As mentioned earlier, GAs mimic the survival-of-the-fittest principle of nature to make a search process.Therefore, GAs are naturally suitable for solving maximization problems. Maximization problems areusually transformed into maximization problem by suitable transformation. In general, a fitness function

F(i) is first derived from the objective function and used in successive genetic operations. Fitness inbiological sense is a quality value which is a measure of the reproductive efficiency of chromosomes. Ingenetic algorithm, fitness is used to allocate reproductive traits to the individuals in the population andthus act as some measure of goodness to be maximized. This means that individuals with higher fitnessvalue will have higher probability of being selected as candidates for further examination. Certain geneticoperators require that the fitness function be non-negative, although certain operators need not have thisrequirement. For maximization problems, the fitness function can be considered to be the same as theobjective function or $F(i) = O(i)$. For minimization problems, to generate non-negative values in all thecases and to reach the relative fitness of individual string, it is necessary to map the underlying naturalobjective function to fitness function form. A number of such transformations is possible. Two commonlyadopted fitness mappings is presented below.

$$F(x) = 1/\{1 + f(x)\} .$$

This transformation does not alter the location of the minimum, but converts a minimization problemto an equivalent maximization problem. An alternate function to transform the objective function to getthe fitness value F(i) as below.

$$\mathcal{F}(i) = V - \frac{O(i)P}{\sum_{i=1}^{P} O(i)}$$

where, O(i) is the objective function value ofith individual, P is the population size and V is a largevalue to ensure non-negative fitness values. The value of V adopted in this work is the maximum value ofthe second term of equation 5 so that the fitness value corresponding to maximum value of the objectivefunction is zero. This transformation also does not alter the location of the solution, but converts a minimization problem to an equivalent maximization problem. The fitness function value of a string isknown as the string fitness.

3. GA operators

The operation of GAs begins with a population of a random strings representing design or decision variables.The population is then operated by three main operators; reproduction, crossover and mutationto create a new population of points. GAs can be viewed as trying to maximize the fitness function, byevaluating several solution vectors. The purpose of these operators is to create new solution vectors byselection, combination or alteration of the current solution vectors that have shown to be good temporarysolutions. The new population is further evaluated and tested till termination. If the termination5criterion is not met, the population is iteratively operated by the above three operators and evaluated.This procedure is continued until the termination criterion is met. One cycle of these operations andthe subsequent evaluation procedure is known as a generation in GAs terminology. The operators aredescribed in the following steps.

3.1.Selection

Reproduction (or selection) is an operator that makes more copies of better strings in a new population.Reproduction is usually the first operator applied on a population. Reproduction selects good stringsin a population and forms a mating pool. This is one of the reason for the reproduction operation tobe sometimes known as the selection operator. Thus, in reproduction operation the process of naturalselection cause those individuals that encode successful structures to produce copies more frequently. Tosustain the generation of a new population, the reproduction of the individuals in the current population isnecessary. For better individuals, these should be from the fittest individuals of the previous population.

There exist a number of reproduction operators in GA literature, but the essential idea in all of themis that the above average strings are picked from the current population and their multiple copies areinserted in the mating pool in a probabilistic manner.

Roulette-Wheel Selection: The commonly-used reproduction operator is the proportionate reproductionoperator where a string is selected for the mating pool with a probability proportional to its fitness. Thus, the ith string in the population is selected with a probability proportional to Fi. Sincethe population size is usually kept fixed in a simple GA, the sum of the probability of each string being

selected for the mating pools must be one. Therefore, the probability for selecting the ith string is

$$p_i = \frac{\mathcal{F}i}{\Sigma \mathcal{F}i}$$

where n is the population size. One way to implement this selection scheme is to imagine a roulette-wheelwithit's circumference marked for each string proportionate to the string's fitness.

3.2.Crossover

A crossover operator is used to recombine two strings to get a better string. In crossover operation,recombination process creates different individuals in the successive generations by combining materialfrom two individuals of the previous generation. In reproduction, good strings in a population areprobabilistically signed a larger number of copies and a mating pool is formed. It is important to notethat no new strings are formed in the reproduction phase. In the crossover operator, new strings arecreated by exchanging information among strings of the mating pool.The two strings participating in the crossover operation are known as parent strings and the resultingstrings are known as children strings. It is intuitive from this construction that good sub-strings fromparent strings can be combined to form a better child string, if an appropriate site is chosen. With arandom site, the children strings produced may or may not have a combination of good sub-strings fromparent strings, depending on whether or not the crossing site falls in the appropriate place. But thisis not a matter of serious concern, because if good strings are created by crossover, there will be morecopies of them in the next mating pool generated by crossover. It is clear from this discussion that theeffect of cross over may be detrimental or beneficial. Thus, in order to preserve some of the good stringsthat are already present in the mating pool, all strings in the mating pool are not used in crossover.When a crossover probability, defined here as pc is used, only 100p_c per cent strings in the populationare used in the crossover operation and 100(1-p_c) percent of the population remains as they are in thecurrent population. A crossover operator is mainly responsible for the search of new strings even thoughmutation operator is also used for this purpose sparingly.

Many crossover operators exist in the GA literature.One site crossover and two site crossover are themost common ones adopted. In most crossover operators, two strings are picked from the mating pool atrandom and some portion of the strings are exchanged between the strings. Crossover operation is doneat string level by randomly selecting two strings for crossover operations. A one site crossover operator isperformed by randomly choosing a crossing site along the string and by exchanging all bits on the rightside of the crossing site.In one site crossover, a crossover site is selected randomly. The portion rightof the selected site of these two strings are exchanged to form a new pair of strings. The new strings arethus a combination of the old strings. Two site crossover is a variation of the one site crossover, exceptthat two crossover sites are chosen and the bits between the sites are exchanged .One site crossover is more suitable when string length is small while two site crossover is suitable forlarge strings. Hence the present work adopts a two site crossover. The underlying objective of crossover is to exchange information between strings to get a string that is possibly better than the parents.

3.3.Mutation

Mutation adds new information in a random way to the genetic search process and ultimately helpsto avoid getting trapped at local optima. It is an operator that introduces diversity in the populationwhenever the population tends to become homogeneous due to repeated use of reproduction and crossover

operators. Mutation may cause the chromosomes of individuals to be different from those of their parentindividuals.Mutation in a way is the process of randomly disturbing genetic information. They operate at thebit level; when the bits are being copied from the current string to the new string, there is probabilitythat each bit may become mutated. This probability is usually a quite small value, called as mutationprobability p_m. A coin toss mechanism is employed; if random number between zero and one is less thanthe mutation probability, then the bit is inverted, so that zero becomes one and one becomes zero. Thishelps in introducing a bit of diversity to the population by scattering the occasional points. This randomscattering would result in a better optima, or even modify a part of genetic code that will be beneficialin later operations. On the other hand, it might produce a weak individual that will never be selectedfor further operations.The need for mutation is to create a point in the neighborhood of the current point, thereby achievinga local search around the current solution. The mutation is also used to maintain diversity in thepopulation. For example, the following population having four eight bit strings may be considered:

01101011
00111101
00010110
01111100.

It can be noticed that all four strings have a 0 in the left most bit position. If the true optimumsolution requires 1 in that position, then neither reproduction nor crossover operator described abovewill be able to create 1 in that position. The inclusion of mutation introduces probability p_m of turning0 into 1.These three operators are simple and straightforward. The reproduction operator selects good stringsand the crossover operator recombines good sub-strings from good strings together, hopefully, to createa better sub-string. The mutation operator alters a string locally expecting a better string. Even thoughnone of these claims are guaranteed and/or tested while creating a string, it is expected that if bad stringsare created they will be eliminated by the reproduction operator in the next generation and if good stringsare created, they will be increasingly emphasized. Further insight into these operators, different waysof implementations and some mathematical foundations of genetic algorithms can be obtained from GAliterature.Application of these operators on the current population creates a new population. This new populationis used to generate subsequent populations and so on, yielding solutions that are closer to theoptimum solution. The values of the objective function of the individuals of the new population areagain determined by decoding the strings. These values express the fitness of the solutions of the new generations. This completes one cycle of genetic algorithm called a generation. In each generation if thesolution is improved, it is stored as the best solution. This is repeated till convergence.

4. Schema Theorem

$$\mathbf{m(H,t+1) \geq m(H,t) * \frac{f(H)}{f} [1 - p_c * \frac{\delta(H)}{l-1} - O(H) * p_m]}$$

The equation above describes how many schema of type H can be observed in the next generation i.e. t+1 given their numbers in this generation i.e. t is m(H,t). According to this theorem highly fit short defining length schemata are propagated from generation to generation.

m(H,t)= no. of strings of schema of type H in a population
f(H)= Average fitness of schema H
f= Average fitness of the population
δ(H) = Defining distance of schema (distance between the 1st and last fixed position)
O(H)= Order of schema (no. of fixed positions in schema)
p_c = Crossover probability
p_m= Mutation probability

IV. Information Theory

1. Information

Let E be some event which occurs with probabilityP(E). If we are told that E has occurred, then wesay that we have received.

$$\mathbf{I(E) = -\log_2 P(E)}\text{bits of information.}$$

Base 2 of log signifies that we are dealing in bits of information. We'll stick with bits, and always assume base 2. Wecan also think of information as amount of "surprise" in E(e.g. P(E) = 1, P(E) = 0)

• Example: result of a fair coin flip ($\log_2 2 = 1$ bit)

• Example: result of a fair die roll ($\log_2 6 = 2.585$ bits)

2. Entropy

A Zero-memory information source S is a source that emits symbols from an alphabet {s1, s2, . . . , sk} with probabilities {p1, p2, . . . , pk},respectively, where the symbols emitted are statistically independent. The average amount of information in observing theoutput of the source S calledas Entropy and mathematically is given by

$$H(s) = \sum_i p_i * I(s_i) = \sum_i p_i * \log_2\left(\frac{1}{pi}\right)$$

Other definitions of the entropy may be stated as follows

1. average amount of information provided per symbol
2. average amount of surprise when observing a symbol
3. uncertainty an observer has before seeing the symbol
4. average number of bits needed to communicate each symbol

(Shannon: there are codes that will communicate these symbols with efficiency arbitrarily close to H(S) bits/symbol;there are no codes that will do it with efficiency < H(S)bits/symbol)

3. Conditional Entropy

If T is alphabet of symbols of a transmitter and R is the alphabet of the receiver then conditional entropy defined by H(T/R) defines the measure of average uncertainty of the transmitted symbol t given the received symbols r. Mathematically it is given by

$$H(T/R) = \sum_i \sum_j p(t,r) \log\left(\frac{p(r)}{p(t,r)}\right)$$

4. Joint Entropy

The joint entropy measures how much entropy is contained in a joint system of two random variables, in our case a receiver R and a transmitter T and represented by H(T,R). Mathematically it is given by

$$\mathbf{H(T,R)=\sum_{i}\sum_{j} p(t,r)\log(t,r)}$$

$$\mathbf{H(T,R)=H(R/T)+H(T)}$$

$$\mathbf{H(T,R)=H(T/R)+H(R)}$$

5. Mutual Information

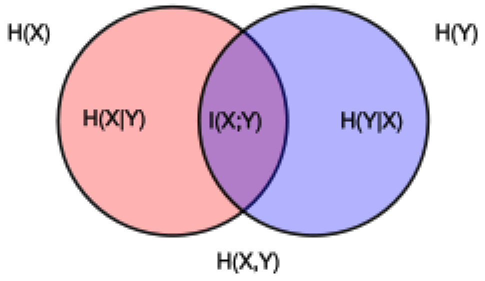
It is the measure of information that one random variable has about the other random variable. It resolves the uncertainty in one random variable after observing the other. Mathematically it is given by

$$\mathbf{I(T,R)=H(T)-H(T/R)}$$

$$\mathbf{I(T,R)=H(R)-H(R/T)}$$

$$\mathbf{I(T,R)=H(T)+H(R)-H(T,R)}$$

$$\mathbf{I(T,R)=H(R)-H(R/T)}$$



6. Entropy and Circuits

Entropy has to be carefully applied to the synthesis ofBoolean functions. Let us assume any two Boolean functions, F1 and F2, and a third F3 which is the one's complement of F2, then F3 ≠ F2. For these complementary functions, H(F2) = H(F3) MI(F1,F2) = MI(F1,F3). Also Mutual Information shows a similar behavior.The implications for Evolutionary Computation are important since careless use of entropy-based measures can prevent the system from attaining convergence. Assume the target Boolean function is T. Then, MI(T, F2) = MI(T, F3), but only one of the circuits implementing F2 and F3 is close to the solution since their Boolean functions are complementary. A fitness function based on mutual information will reward both circuits with the same value, but one is better than the other, Things could get worse as evolution progresses because the mutual information increases when the circuits are closer to the solution, but in fact, two complementary circuits are then given larger rewards. The scenario is one in which the population is driven by two equally strong attractors, hence convergence is never reached.Mutual information is not an invariant measure between random variables because it contains the marginal entropies. Normalized Mutual Information is a better measure of theprediction that one variable can do about the other. Normalized Mutual Information is given by

$$\textbf{NMI} = \frac{\boldsymbol{H(R)+H(T)}}{\boldsymbol{H(T,R)}}$$

V. Binary Decision Diagrams (BDD)

1. Introduction

With the ever increasing complexity of digital functionsand systems, the researcher who is charged withtheir analysis, testing, and implementation is faced with avery real "description" dilemma. On the one hand, he has athis disposal a variety of sophisticated design languageswhich can provide concise functional descriptions of thedevice or system with which he is concerned. However, when

he attempts to use such descriptions in any sort of formalanalysis procedure, he typically discovers that their veryconciseness virtually precludes any detailed logical investigation.On the other hand, when he turns to those descriptionswhich are amenable to extensive analysis, he finds thatthese take the form of truth tables, Boolean equations,Karnaugh maps, etc.-all of which have the unpleasantproperty of growing exponentially with the number ofvariables involved. What he would like to have would be aconcise, "implementation-free" description which could stillyield meaningful results about the logical structure andtesting requirements of the function involved. This paperwill explore one possible approach to bridging this "descriptiongap."The general idea will be to define a digital function interms of a "diagram" which tells the user how to determinethe output value of the function by examining the values ofthe inputs. We shall begin by describing these diagrams andshowing how they may be derived for various digital devices.

2. Implementation

This technique is based on the Shannon's expansion

$$\mathbf{f(A,B,C..)=!Af_0(0,B,C,..) + Af_1(1,B,C..)}$$

Consider the switching function,

$$f= A+(!B.C)$$

and assume we are interested in defining a procedure fordetermining the binary value offgiven the binary values ofA, B, and C. One way to do this would be to begin by lookingat the value of A. If A = 1, thenf= 1 and we are finished. IfA = 0, we look at B. If B = 1, thenf= 0 and again we arefinished. Otherwise, we look at C and its value will be thevalue offFig. 1 shows a simple diagram ofthis procedure. We enterat the node indicated by the arrow and then simply proceeddownward through the diagram, noting at each node thevalue of its variable and then taking the indicated branch.When a 0 or 1 value is reached, this gives the value offandthe process ends. There is little difficulty in confirming that the diagram doesindeed describe a procedure for finding the value of theindicated function. We shall refer to these diagrams as binarydecision diagrams.

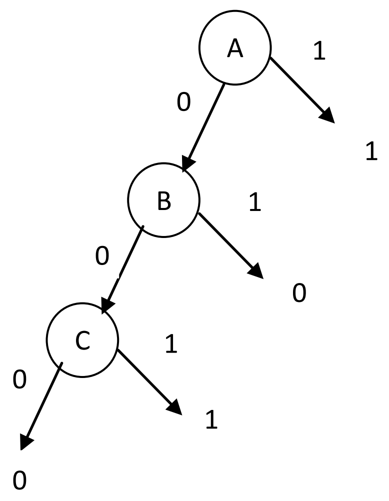
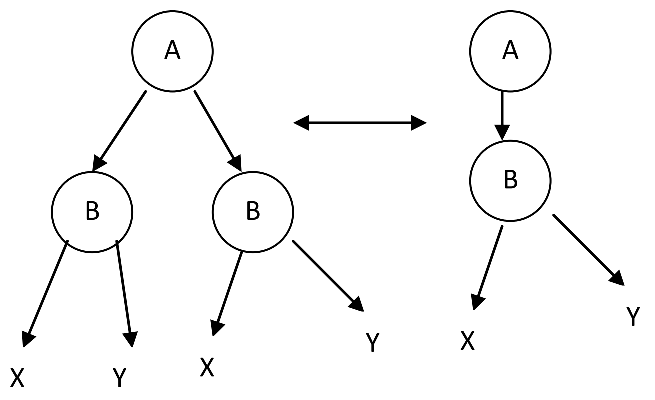
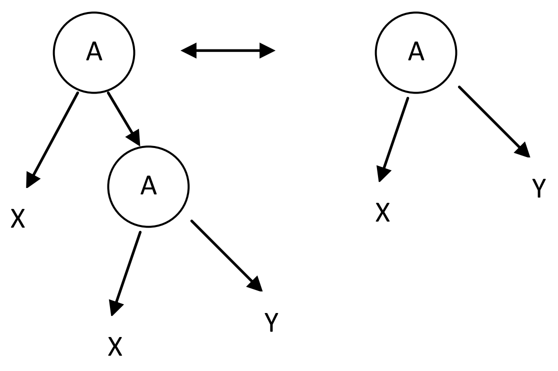


Figure 1

There can be redundant nodes in the BDD which can be pruned by followingrules which are described below with the help of diagrams.



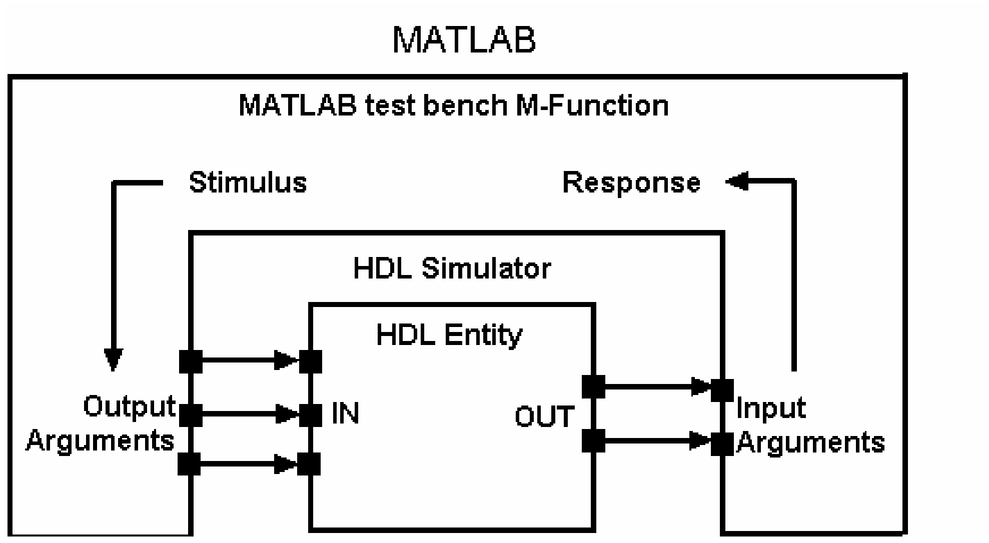


Rule 1 Rule 2

VI. MATLAB EDA Simulator Link

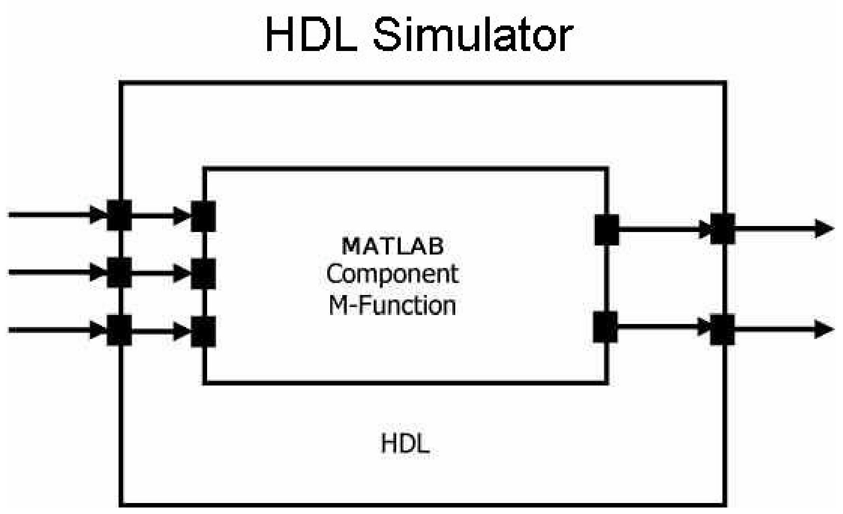
1. Writing Test Bench in MATLAB for HDL Combinational Logic

TheEDA Simulator LinksoftwareprovidesameansforverifyingHDLmoduleswithintheMATLABenvironment. YoudosobycodinganHDLmodeland aMATLABfunctionthatcansharedatawiththeHDLmodel. *MATLABtestbench*functionsletyou verify theperformanceoftheHDL model,orofcomponentswithinthemodel. Atestbenchfunctiondrives valuesontosignalsconnectedtoinputportsofanHDLdesignundertestand receivessignalvaluesfromtheoutputportsofthemodule.Thefollowingfigureshowshowa MATLABfunctionwrapsaroundand communicateswiththeHDLsimulatorduringa testbenchsimulation session.



2. HDL Cosimulation Using MATLAB Component Function

TheEDA Simulator LinksoftwareprovidesameansforvisualizingHDLcomponents withinthe MATLABenvironment.You dosobycodinganHDLmodelandaMATLABfunctionthatcansharedatawiththeHDLmodel. *MATLABcomponent* functionssimulatethebehavior ofcomponentsinthe HDLmodel.Astubmodule(providingportdefinitions only)intheHDLmodel passesitsinputsignalstotheMATLABcomponentfunction. TheMATLAB componentprocessesthisdataandreturnstheresultstotheoutputsofthe stubmodule. AMATLABcomponenttypicallyprovidessomefunctionality (suchasafilter)thatisnotyetimplementedintheHDLcode. Thefollowingfigureshow showanHDLsimulatorwrapsaroundaMATLAB



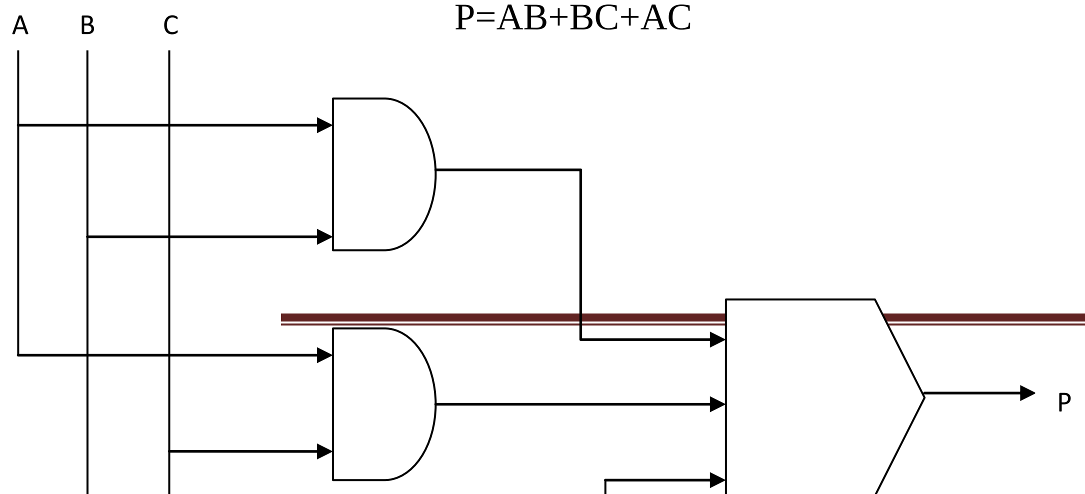
component functionandhowMATLABcommunicates withtheHDL simulatorduringacomponentsimulationsession.

VII. Circuit Evolution

1. 3,2 Majority Logic Circuit

A 3,2 majority circuit is a combinational circuit that has one output (P) which is equal to 1 if the number of 1s on the input are greater than the number of 0s and the output (P) equals 0 if the number of 1s on the input are less than the number of 0s . The switching function, the logic diagram and the truth table are given below.

$$P=AB+BC+AC$$



Inputs			Outputs
A	B	C	P
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 1

shows the truth table of the 3,2 Majority function

Table -1

2. Universal Logic Module for 3 Variables

The Universal Logic module for 3 variables is capable of implementing any combinational logic function out of a possible combination of 2^8=256 for 3 input variables. It has 3 inputs**a, b &c**and one output **f,** in addition there is **cntl**input 29 bits wide which is used to programme the module.

The module consists of 7, 2x1 multiplexers, the control inputs to each multiplexers can be programmed through cntl input of the module. Any switching function of n variables can be programmed using 2^n-1 2x1 multiplexers. The module is written in Verilog HDL and the code for the module is available in Annexure-. The module has been designed deliberately using only 2x1 multiplexers as in the VLSI system design where the emphasis is to reduce the whole manufacturing cost rather than reduce the number of components used. It is common therefore to replicate the same unit as many times as possible, although it may lead to larger number of gates. Our interest in VLSI system design leads us to restate the circuit design problem in such a way that the issues mentioned previously are taken into account. Figure 1 shows the block diagram of the module and figure 2 shows the internal structure of the module.

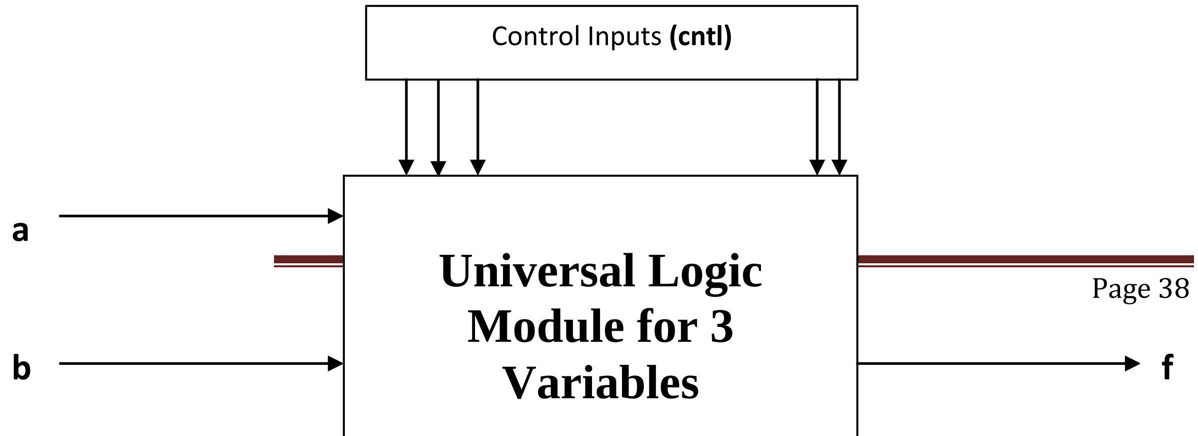


Figure -1

In figure 2, the internal structure of the module, all the control inputs of the muxes are programmable, they can have either of the values 0,1,a,~a,b,~b,c,~c depending upon the control input **"cntl"**. The inputs to the muxes 1,2,3 and 4 can have value either 0 or 1 again depending upon the control input **"cntl"**. The control muxes are not shown in the figure 2. Table 2 shows the relation of input values to muxes 1,2,3 and 4 vis a vis the control input. The input values can either be a '0' or a '1' depending upon the implementation of the function. Table 3 shows the relation between the control input of every 2x1 mux and the vis a vis control input.

Control Input	Output
0	0
1	1

Table 1

Control Input	X
000	0
001	1
010	a
011	!a
100	b
101	!b
110	c
111	!c

Table 2

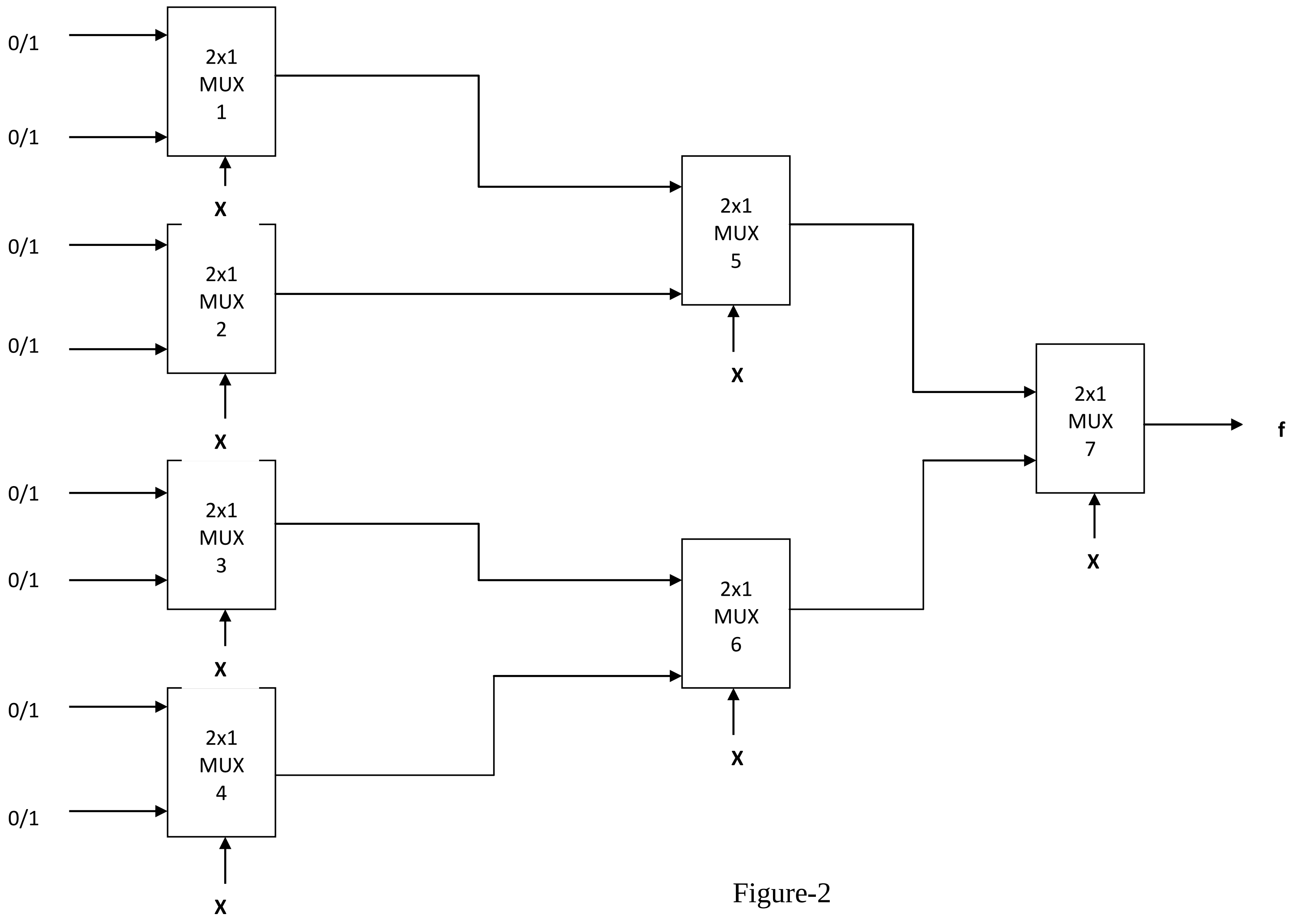

0/1
0/1
0/1
0/1
0/1
0/1
0/1
0/1
2x1 MUX 1
2x1 MUX 2
2x1 MUX 3
2x1 MUX 4
2x1 MUX 5
2x1 MUX 6
2x1 MUX 7
X
X
X
X
X
X
X
f

Figure-2

3. Fitness Function

Information Theory measures were used for the calculation of the fitness function. Conditional Entropy can be used as one of the measures for the calculation of the fitness functions, but as stated earlier It has a big drawback that it rewards both the function and complement of the function equally, we may end up with a circuit that does not meet our requirement. Instead we can Normalised Mutual Information (NMI) and function that uses Conditional Entropy that does not reward the complement of the function as can be seen in equation (3).

$$fitness= (Length(T) - Hamming(T,C)) \times NMI(T,C) \qquad (1)$$

$$fitness1=\sum_i \frac{fitness}{NMI(Ai,C)} (2)$$

$$fitness2=(Length(T) - Hamming(T,C))x(10-H(T/C) \qquad (3)$$

Where T is the actual output and C is the simulated output. A_i are the attributes (variables) of the target function. To evolve the required combinational circuit both (2) and (3) have to be maximized. As the tool used by us for GA optimization is able to achieve global minimum we have to convert both equation (2) and (3) to using a transformation. Thus we have a function given by (4) used as a fitness function.

$$f=\frac{1}{1+fitness1} (4)$$

4. GA Parameters

Table 3 shows the parameters used in GA for evolution of the 3,2 majority function circuit. The tool used for optimization was Global Optimization Toolbox of MATLAB.

Parameters	Value
Population Size	150
No. of Generations	25
Crossover Rate	0.8
Crossover Type	Two Point
Mutation Rate	0.2
Mutation Type	Uniform
Scaling	Rank
Elitism	2
Selection	Stochastic Uniform

Table 3

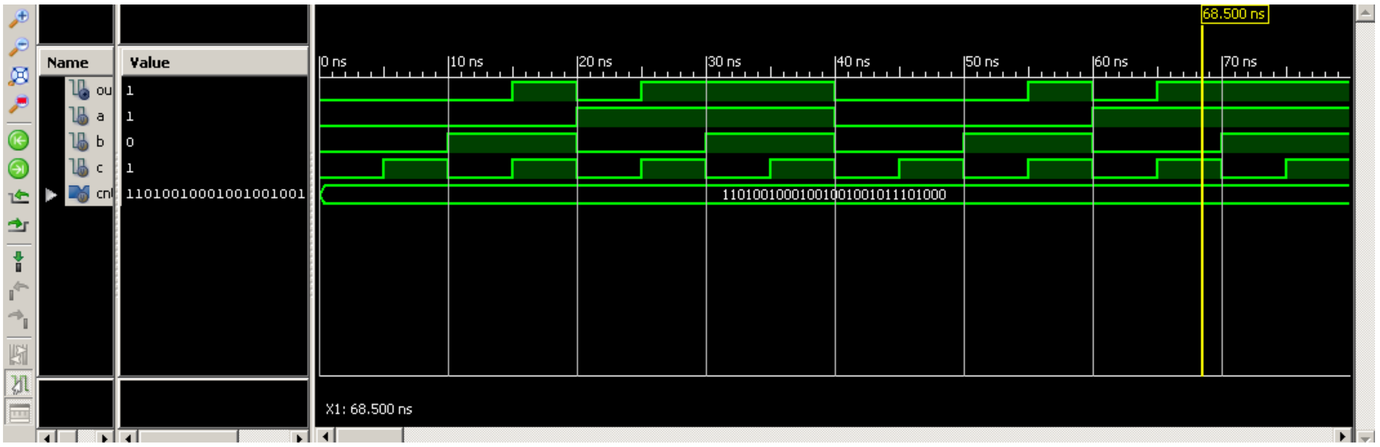
VIII. Results and Conclusion

1. Verification of the Evolved 3,2 Majority Function

The 3,2 majority function circuit was evolved using programmable universal module for 3 variables having a programming input of 29 bits labeled as “cntl”. These 29 bits were used by the GA to evolve the 3,2 majority function. The module itself constituted a mux tree of 7, 2x1 multiplexers, with control inputs of either 0,1,a, !a, b, !b c, !c depending upon the control input “cntl”. The inputs to level 3 mux can take values of either a ‘0’ or a ‘1’ again depending upon the control input “cntl”.

The GA string which is nothing but the control input “cntl” is coded in the following manner. The chromosome consists of a string of 29 bits which form the control input “**cntl**” to the universal logic module for 3 variables. Bits from 0 to 7 decide what inputs to be fed to inputs of level 3 multiplexers, the inputs to the muxes can have a value of either a ‘0’ or a ‘1’ only. Bits from 8 to 19 decide what variable is to be fed as control inputs to muxes at level 3,similarly the bits from 20 to 25 decide the control input for level 2 muxes and bits 26 to 28 decide control input for level 1 mux. The relation, for what input combination what control signal is chosen has already been described table 2 of the previous chapter.

The GA was able to identify a number of input combinations for which the output met the desired criteria. The one describe here uses minimum number of muxes which can be considered as the optimum solution to the problem of evolving 3,2 majority function. The figure 1 below of the simulated clearly shows that the objectives have been met and circuit has been evolved.



Name	Value
ou	1
a	1
b	0
c	1
cnt	11010010001001001001

```
11010010001001001001011101000
```

Figure 1

2. Evolved and Optimized Module

The evolved module with inputs to stage 3 and control stages to all the 3 stages is shown in figure 2. The optimized module with thewith minimum number of multiplexers is shown in figure 3.

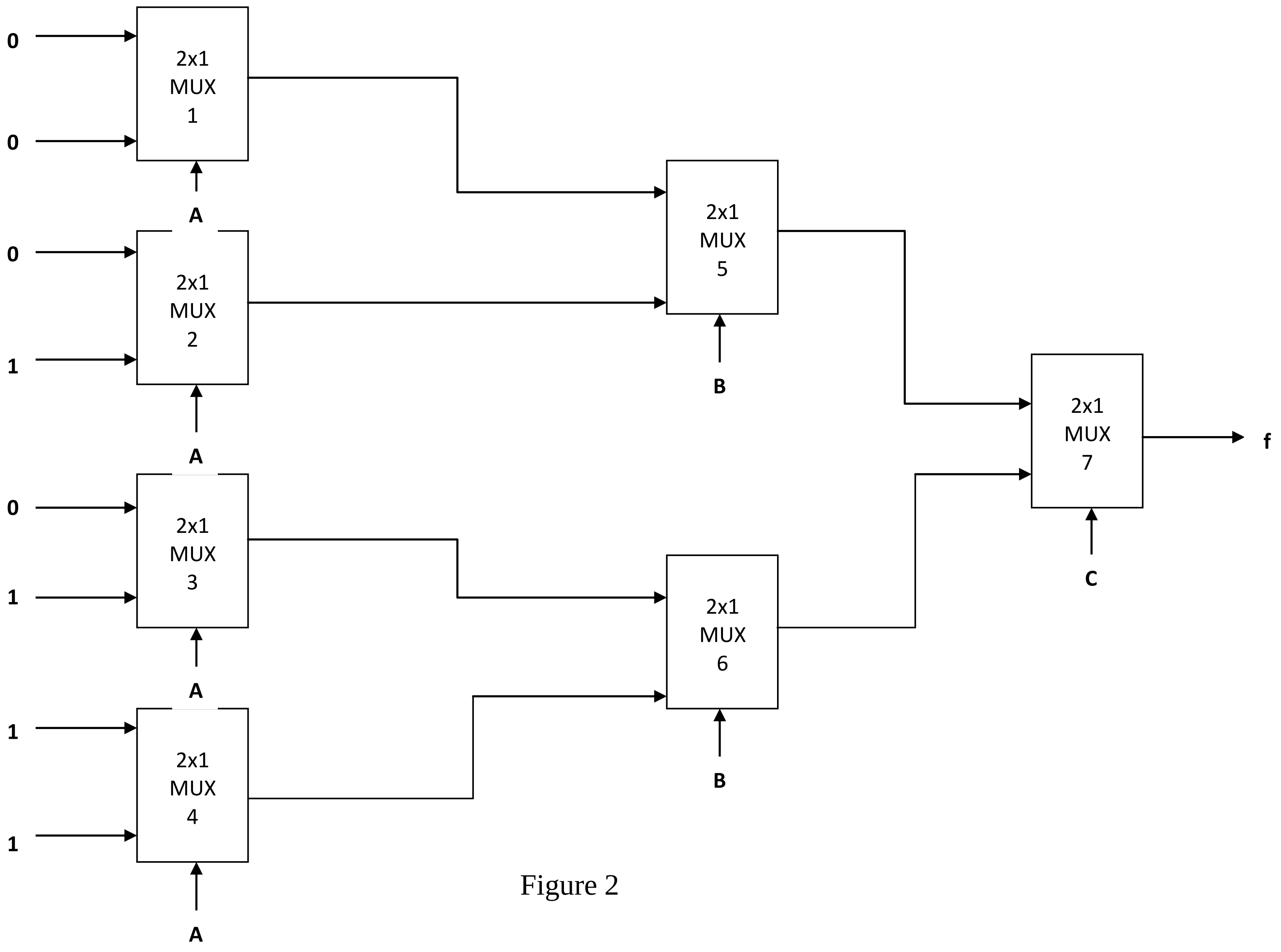

0
0
2x1 MUX 1
A
0
1
2x1 MUX 2
A
0
1
2x1 MUX 3
A
1
1
2x1 MUX 4
A
2x1 MUX 5
B
2x1 MUX 6
B
2x1 MUX 7
C
f

Figure 2

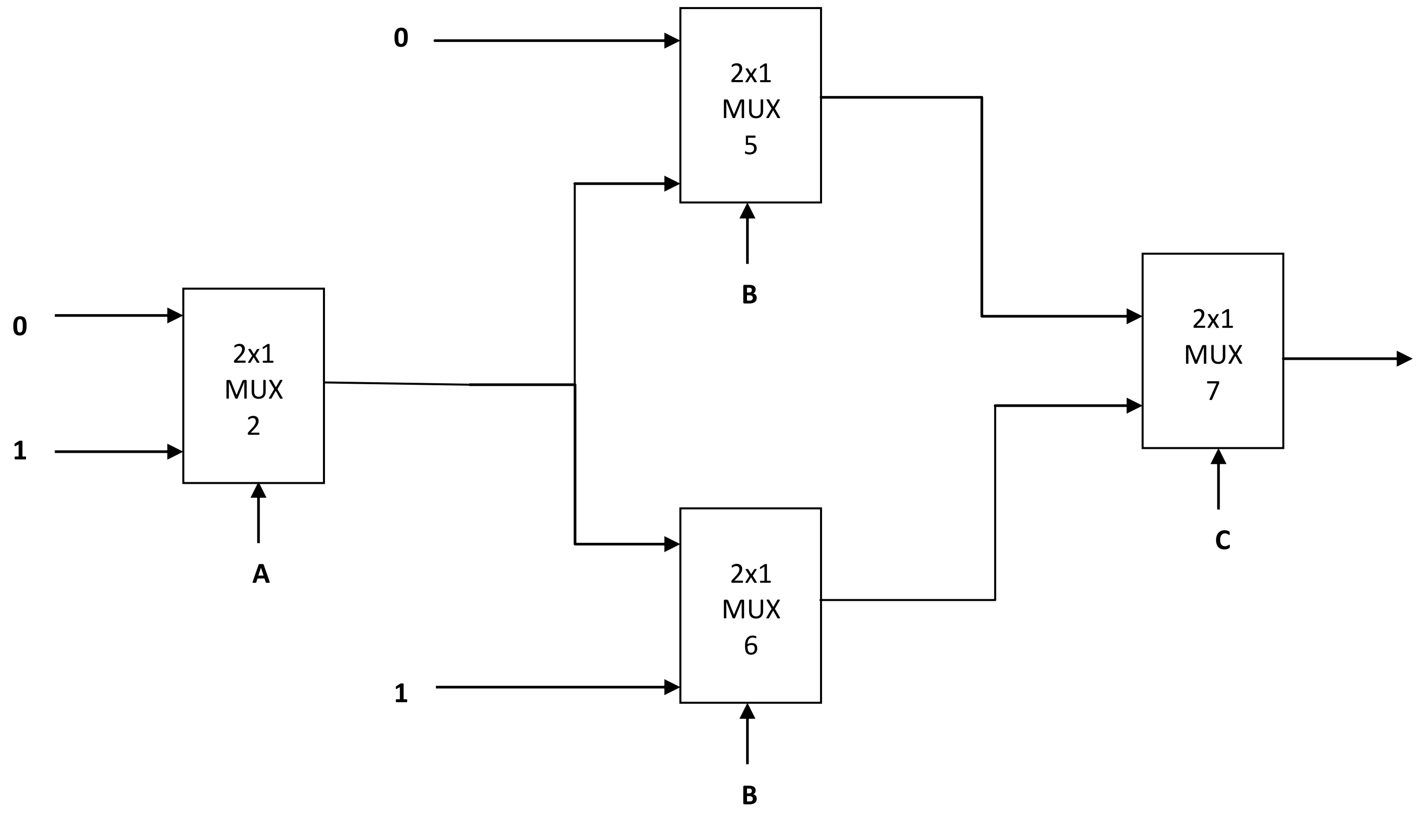


Figure 3

It is clearly visible from the figure 3 that the requirement for the number of multiplexers has come down from 7 to 4. It will be shown in the next section using ROBDD that this is the canonical form of the circuit and hence optimized.

3. Binary Decision Diagrams (BDD) and Reduced Ordered Binary Decision Diagrams (ROBDD)

In order to verify our conclusion we took the help of BDD and ROBDD. The BDD and ROBDD for the 3,2 Majority functions are presented in figure 4 and 5 respectively.

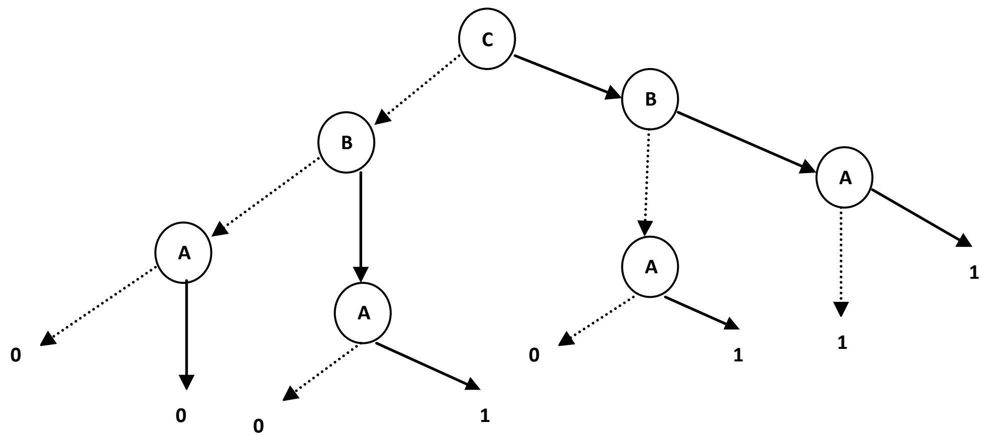


Figure 4

The dashed lines show '0' as the input and solid lines show '1' as the input. The BDD is in agreement with the evolved circuit of figure 2. The ROBDD is shown in the next figure with the order being C<B<A.

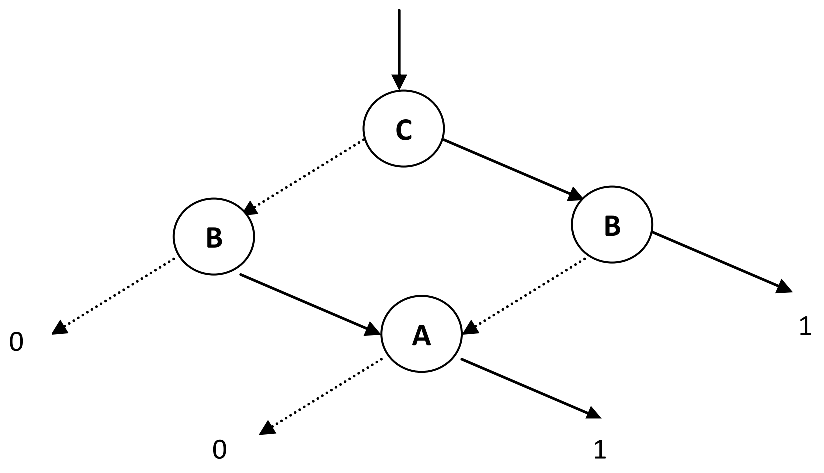


Figure 5

From figure 5 it can be clearly seen that the reduced and optimized circuit of the figure 3 is in agreement to that of ROBDD.

4. GA Convergence

Figure 6 shows the convergence of the GA with generations and the mean and the average value of the fitness function for a given generation. It can be seen that the value of the fitness function converges to 0.0185 after generation 9 till 25.

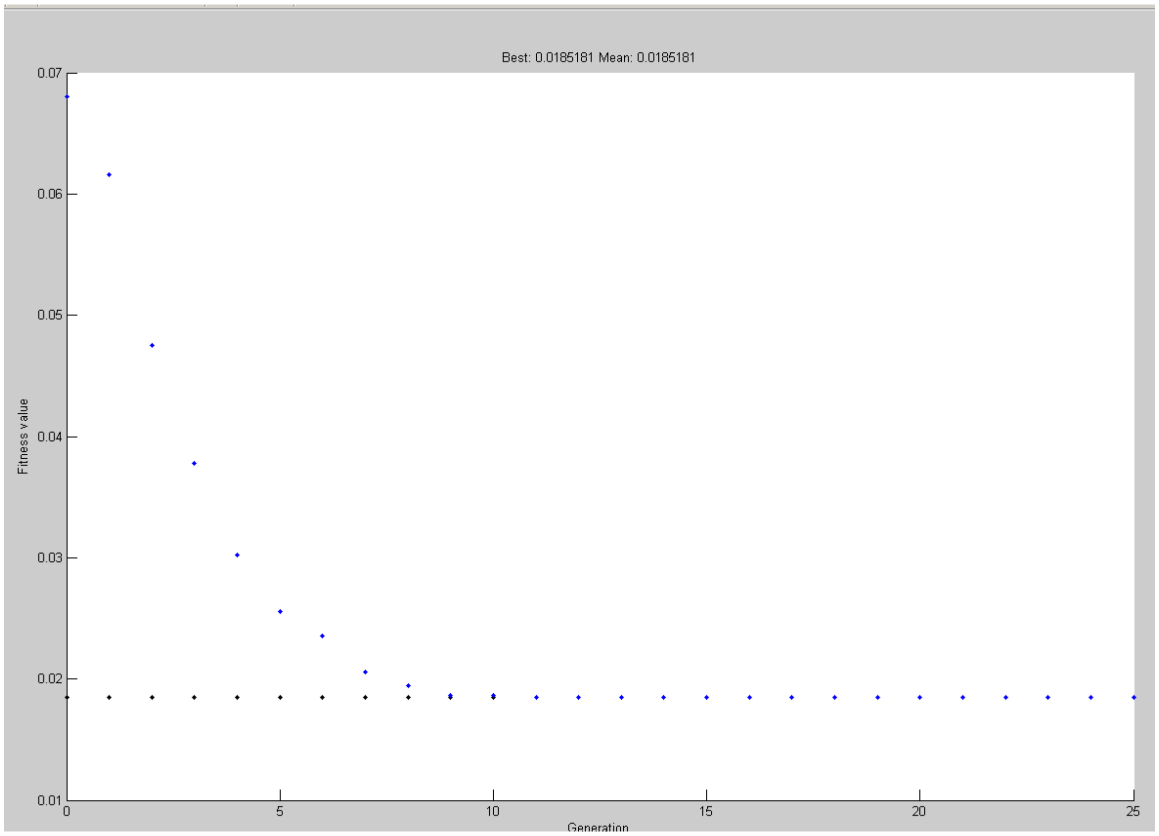


Figure 6

Figure 7 shows the best, worst and the mean values of the fitness function over the entire range of the 25 generations. Again form this graph it is very clear that after generation 10 the GA has converged to one single value of the fitness function which is the global minimum for the fitness function having a value of 0.0185 and leads to the evolution of the combinational circuit.

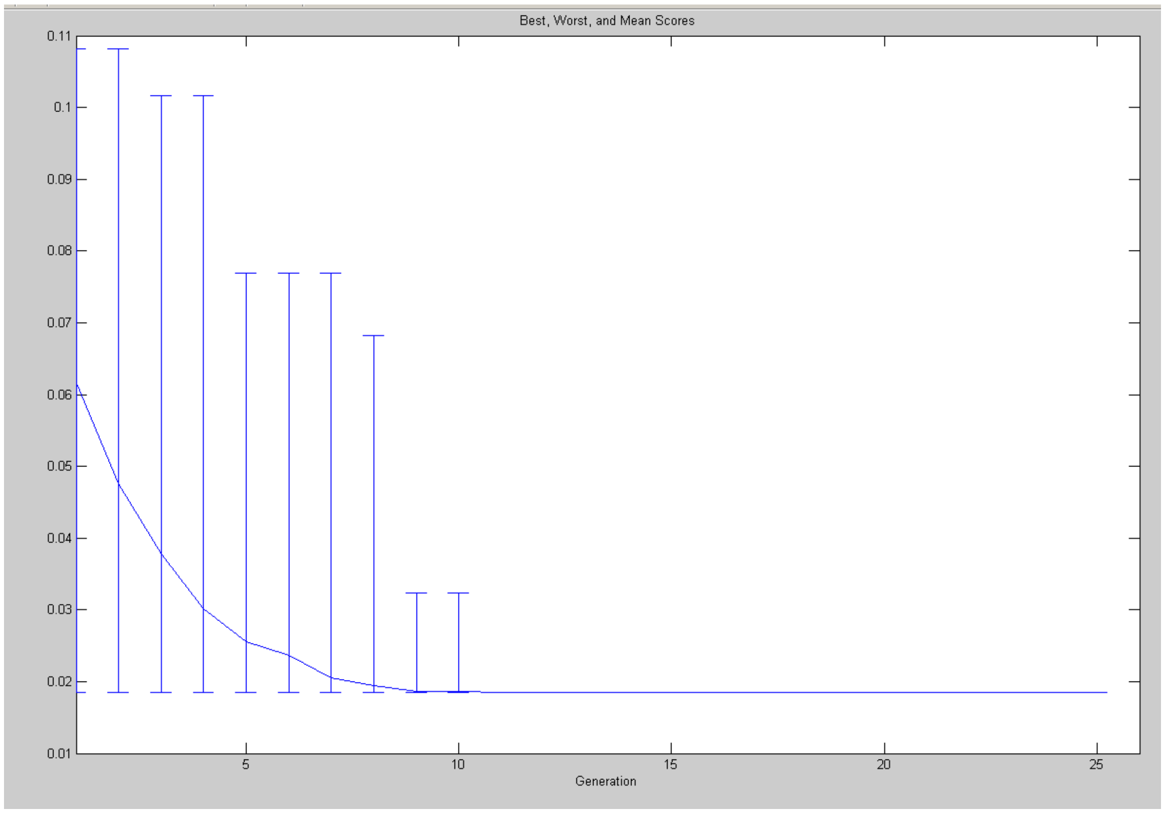


Figure 7

5. Conclusion

In this project we have been able to evolve a digital combination circuit using GA algorithm as a search tool for optimum solution in a very large search space. We have shown that it is possible to incorporate information theory indices like Normalized Mutual Information for the creation of fitness function. The solution so found was compared to the optimum solution which was determined by ROBDD and has least number of elements i.e. 2x1 muxes.

6. Future Work

We plan to take this work further and carry out the evolution to other more complex arithmetic combinational circuits such as multipliers. We also plan to carryout work on sequential circuits such as sequence detector.

Appendix – 1

Verilog Module for Universal Logic Module for 3 Variables

File Name :-majority_mux.v

```
modulemajority_mux(cntl,a,b,c,out);
input [28:0] cntl;
input a;
input b;
input c;
output out;
reg out;
reg [20:0] w;
always @(a,b,c) begin
if (cntl[10:8]== 3'b000)
w[8]=1'b0;
else if (cntl[10:8]== 3'b001)
w[8]=1'b1;
else if (cntl[10:8]== 3'b010)
w[8]=a;
else if (cntl[10:8]==3'b011)
w[8]=~a;
else if (cntl[10:8]== 3'b100)
w[8]=b;
else if (cntl[10:8]== 3'b101)
w[8]=~b;
else if (cntl[10:8]==3'b110)
w[8]=c;
else if (cntl[10:8]== 3'b111)
w[8]=~c;

if (cntl[13:11]== 3'b000)
w[8]=1'b0;
else if (cntl[13:11]== 3'b001)
w[9]=1'b1;
else if (cntl[13:11]== 3'b010)
w[9]=a;
else if (cntl[13:11]== 3'b011)
w[9]=~a;
else if (cntl[13:11]== 3'b100)
w[9]=b;
else if (cntl[13:11]== 3'b101)
w[9]=~b;
else if (cntl[13:11]== 3'b110)
w[9]=c;
else if (cntl[13:11]== 3'b111)
w[9]=~c;

if (cntl[16:14]== 3'b000)
```

```
w[10]=1'b0;
else if (cntl[16:14]== 3'b001)
w[10]=1'b1;
else if (cntl[16:14]== 3'b010)
w[10]=a;
else if (cntl[16:14]== 3'b011)
w[10]=~a;
else if (cntl[16:14]== 3'b100)
w[10]=b;
else if (cntl[16:14]== 3'b101)
w[10]=~b;
else if (cntl[16:14]== 3'b110)
w[10]=c;
else if (cntl[16:14]== 3'b111)
w[10]=~c;

if (cntl[19:17]== 3'b000)
w[11]=1'b0;
else if (cntl[19:17]== 3'b001)
w[11]=1'b1;
else if (cntl[19:17]== 3'b010)
w[11]=a;
else if (cntl[19:17]== 3'b011)
w[11]=~a;
else if (cntl[19:17]== 3'b100)
w[11]=b;
else if (cntl[19:17]== 3'b101)
w[11]=~b;
else if (cntl[19:17]== 3'b110)
w[11]=c;
else if (cntl[19:17]== 3'b111)
w[11]=~c;

if (cntl[22:20]== 3'b000)
w[16]=1'b0;
else if (cntl[22:20]== 3'b001)
w[16]=1;
else if (cntl[22:20]== 3'b010)
w[16]=a;
else if (cntl[22:20]== 3'b011)
w[16]=~a;
else if (cntl[22:20]== 3'b100)
w[16]=b;
else if (cntl[22:20]== 3'b101)
w[16]=~b;
else if (cntl[22:20]== 3'b110)
w[16]=c;
else if (cntl[22:20]== 3'b111)
w[16]=~c;

if (cntl[25:23]== 3'b000)
w[17]=1'b0;
else if (cntl[25:23]== 3'b001)
w[17]=1'b1;
else if (cntl[25:23]== 3'b010)
w[17]=a;
else if (cntl[25:23]== 3'b011)
```

```
w[17]=~a;
else if (cntl[25:23]== 3'b100)
w[17]=b;
else if (cntl[25:23]== 3'b101)
w[17]=~b;
else if (cntl[25:23]== 3'b110)
w[17]=c;
else if (cntl[25:23]== 3'b111)
w[17]=~c;

if (cntl[28:26]== 3'b000)
w[20]=1'b0;
else if (cntl[28:26]== 3'b001)
w[20]=1'b1;
else if (cntl[28:26]== 3'b010)
w[20]=a;
else if (cntl[28:26]== 3'b011)
w[20]=~a;
else if (cntl[28:26]== 3'b100)
w[20]=b;
else if (cntl[28:26]== 3'b101)
w[20]=~b;
else if (cntl[28:26]== 3'b110)
w[20]=c;
else if (cntl[28:26]== 3'b111)
w[20]=~c;

if (cntl[0]==1'b0)
w[0]=1'b0;
else
w[0]=1'b1;

if (cntl[1]==1'b0)
w[1]=1'b0;
else
w[1]=1'b1;

if (cntl[2]==1'b0)
w[2]=1'b0;
else
w[2]=1'b1;

if (cntl[3]==1'b0)
w[3]=1'b0;
else
w[3]=1'b1;

if (cntl[4]==1'b0)
w[4]=1'b0;
else
w[4]=1'b1;

if (cntl[5]==1'b0)
w[5]=1'b0;
else
w[5]=1'b1;
```

```
if (cntl[6]==1'b0)
w[6]=1'b0;
else
w[6]=1'b1;

if (cntl[7]==1'b0)
w[7]=1'b0;
else
w[7]=1'b1;

if (w[8]==0)
w[12]=w[0];
else
w[12]=w[1];

if (w[9]==0)
w[13]=w[2];
else
w[13]=w[3];

if (w[10]==0)
w[14]=w[4];
else
w[14]=w[5];

if (w[11]==0)
w[15]=w[6];
else
w[15]=w[7];

if (w[16]==0)
w[18]=w[12];
else
w[18]=w[13];

if (w[17]==0)
w[19]=w[14];
else
w[19]=w[15];

if (w[20]==0)
out=w[18];
else
out=w[19];
end
endmodule
```

Appendix – 2

Testbench File for Veirlog Module in MATLAB

File Name :-majority_mux.m

```
functionmajority_mux(obj)
global s1;
global x1;
global y;
obj.tnext=obj.tnow+5e-9;
s1=s1+1;
if (strcmp(obj.simstatus,'Init'))
obj.portvalues.cntl=dec2mvl(x1,29);
obj.userdata.sum=int16.empty;
y=int16.empty;
end
if (strcmp(obj.simstatus,'Running'))
mvl2dec(obj.portvalues.out);
y(s1-1)=mvl2dec(obj.portvalues.out);
end
if s1==8
obj.portvalues.a= dec2mvl(1,1);
obj.portvalues.b=dec2mvl(1,1);
obj.portvalues.c=dec2mvl(1,1);



return
end
if s1==7
obj.portvalues.a= dec2mvl(1,1);
obj.portvalues.b=dec2mvl(1,1);
obj.portvalues.c=dec2mvl(0,1);

return
end
if s1==6
```

```
obj.portvalues.a= dec2mvl(1,1);
obj.portvalues.b=dec2mvl(0,1);
obj.portvalues.c=dec2mvl(1,1);



return
end
if s1==5
obj.portvalues.a= dec2mvl(1,1);
obj.portvalues.b=dec2mvl(0,1);
obj.portvalues.c=dec2mvl(0,1);

return
end
if s1==4
obj.portvalues.a= dec2mvl(0,1);
obj.portvalues.b=dec2mvl(1,1);
obj.portvalues.c=dec2mvl(1,1);

return
end
if s1==3
obj.portvalues.a= dec2mvl(0,1);
obj.portvalues.b=dec2mvl(1,1);
obj.portvalues.c=dec2mvl(0,1);

return
end
if s1==2
obj.portvalues.a= dec2mvl(0,1);
obj.portvalues.b=dec2mvl(0,1);
obj.portvalues.c=dec2mvl(1,1);

return
end
if s1==1
obj.portvalues.a= dec2mvl(0,1);
obj.portvalues.b=dec2mvl(0,1);
obj.portvalues.c=dec2mvl(0,1);

return
end

if s1==9
obj.tnext=[];
end
end
```

Appendix -- 3

Evaluation Function File

File Name:-majority.m

```
function f=majority(x)
global y;
global r;
global z;
global s1;
global f;
global f1;
global x1;
s1=0;
globaltclcmd;
a=[0;0;0;0;1;1;1;1];
b=[0;0;1;1;0;0;1;1];
c=[0;1;0;1;0;1;0;1];
r=[0;0;0;1;0;1;1;1];
r=int16(r);
x1=x;
commInfo = hdldaemon;
tclcmd = { ...
'vsimmatlabmajority_mux -t 1ns -novopt ;', ...
['matlabtb /majority_mux  -mfuncmajority_mux.m -use_instance_obj  ;'], ...
'run 100;',...
'quit -f',...
};
% Launch HDL simulator for use with MATLAB
vsim('tclstart',tclcmd,'runmode','Batch');
pause(8);
y=int16(y);
y=y';
```

```
f1=[(8-sum(abs(r-y)))*(entropy(r)+entropy(y))/(entropy(r)+entropy(y)-
mutualinformation(r,y))];
f2=f1*[(entropy(y)+entropy(a))/(entropy(y)+entropy(a)-
mutualinformation(a,y))+...
    (entropy(y)+entropy(b))/(entropy(y)+entropy(b)-
mutualinformation(b,y))+...
    (entropy(y)+entropy(c))/(entropy(y)+entropy(c)-mutualinformation(c,y))];
f=(1/(1+f2));
if (y==r)
done=dec2mvl(x1,29);
fprintf('correct value = %s\n',done);
fid = fopen('values.txt', 'a');
fprintf(fid, 'VectorXVectorYFunctionValue\n');
fprintf(fid, '%s  %d %d %d %d %d %d %d %d   %6.4f\n',done,y,f);
fclose(fid);
end
```

References

[1] Digital Design by Mano, M. M

[2] www.mathworks.com

[3] EDA Simulator Link User Guide

[4] Advance Digital Design with Verilog HDL by Michael D. Cilett

[5] Verilog HDL Design and Modeling by Joseph Cavanagh

[6] Genetic Algorithm in Search, Optimization and Machine Learning by David Goldberg

[7] Mutual Information-based Fitness Functions for Evolutionary Circuit Synthesis by Arturo Hemindez Aguirre and Carlos A. Coello Coello.

[8] Evolutionary Synthesis of Logic Functions using Multiplexers by Arturo Hemindez Aguirre ,Bill P Buckles and Carlos A. Coello Coello.

[9] Gate level synthesis of Boolean Functions using Binary Multiplexers and Genetic Programming by Arturo Hemindez Aguirre ,Bill P Buckles and Carlos A. Coello Coello.

[10] Information Theory Method for Flexible Network Synthesis by V. Cheushev, S. Yanushkevich, V. Shmerko, *C.* Moraga and J. Kolodziejczyk

[11] Evolutionary Algorithms and Their Use in the Design of Sequential Logic Circuits by B.ALI and A.E.A.ALMAINI

[12] Application of Design Style in EvolutionaryMulti-Level Networks Synthesis by TadeuszLuba, Claudio Moraga, Svetlana Yanushkevich, VladShmerko, and Joanna Kolodziejczyk

[13] Binary Decision Diagrams by Sheldon B. Akers

[14] Graph-Based Algorithms for Boolean Function Manipulation by Randal E Bryant.

[15] Designing Electronic Circuits Using Evolutionary Algorithms.Arithmetic Circuits: A Case Study by J. F. Miller, P.Thomson, T. Fogarty

[16] Algorithm For Logic-Circuit Synthesis By Using Multiplexers By DG Whitehead

[17] Evolution of Digital Circuits with Variable Layout by Tatiana Kalganov, J. F. MILLERand T. FOGARTY

[18] Universal Logic Modules and their Application by STEPHEN S. YAU and CALVIN K. TANG