Multi-Objective BDD Optimization for RRAM based Circuit Design

Saeideh Shirinzadeh*, Mathias Soeken†, Rolf Drechsler*‡
*Department of Mathematics and Computer Science, University of Bremen, Germany
†Integrated Systems Laboratory, EPFL, Lausanne, Switzerland
‡Cyber-Physical Systems, DFKI GmbH, Bremen, Germany
{saeideh,drechsle}@cs.uni-bremen.de, mathias.soeken@epfl.ch

Abstract—Resistive switching property enables various promising applications such as design of non-volatile in-memory computing devices which has attracted high attention to Resistive Random Access Memories (RRAMs). In this work, we present a multi-objective BDD optimization approach for RRAM based logic circuit design. Dissimilar to classical BDD optimization, evaluating the cost metrics of the circuits in this case does not only depend on the number of BDD nodes but is more advanced. We have utilized a non-dominated sorting genetic algorithm for bi-objective BDD optimization with respect to the number of required RRAMs and computational steps addressing the area and delay of the resulting circuits, respectively. The algorithm allows preference to one of the objectives if it is of higher significance. Experimental results show that the proposed multi-objective genetic algorithm achieves considerable reduction in both aforementioned criteria in comparison with an existing approach.

I. INTRODUCTION

Binary Decision Diagrams (BDDs) are graph-based data structures and are proven to be efficient for representing and manipulating Boolean functions. Electronic design automation and formal verification greatly benefit from BDDs in many applications. BDD optimization approaches especially node minimization techniques have been widely used in these applications. BDDs can be transferred directly to circuits by mapping each node to a multiplexer which makes the number of nodes the main cost metric. Nonetheless, there are few approaches which optimize BDDs with respect to other criteria such as the number of paths or expected and average path length [1]. Hence, in the majority of applications BDD optimization is classically defined as minimizing the number of BDD nodes which can be directly mapped to multiplexers to synthesize a circuit.

The abrupt switching capability of an oxide insulator sandwiched by two metal electrodes was known from 1960s, but it did not come into interest for several decades until feasible device structures were proposed [2]. Nowadays, a variety of two-terminal devices based on resistance switching property exist which use different materials. In [3], Resistive Random Access Memories (RRAMs) were suggested as a physical implementation for the theory of memristors proposed in [4]. Although some researchers argued differences between memristor and RRAM, in this paper we use RRAM to refer to a generic resistive switching memory.

High scalability of RRAMs [2] makes it possible to implement ultra dense resistive memory arrays in hybrid nano/CMOS technology [5]. Such hybrid architectures using memristive devices are of high interest for their possible applications in non-volatile memory design, digital and analog programmable systems, and neuromorphic computing structures [6].

In [7], it was shown that Material Implication (IMP) can be executed by resistive switches to build logic circuits. In the same work a NAND gate was proposed that consists of three RRAMs and thus enables to realize any Boolean function. This allows advanced computer architectures different from classical von Neumann architectures by providing memories able to perform computing operations [8]. So far, researchers have proposed various resistive logic circuits based on IMP operators. An RRAM based 2-to-1 multiplexer (MUX) containing six RRAMs was proposed in [6] that requires seven operations. In [9], a similar structure but more efficient in the number of RRAMs and operations was used for synthesis of Boolean functions using BDDs.

Although RRAM based implication logic is sufficient to express any Boolean function, the number of required computational steps to synthesize a given function is a real drawback [10]. So far, few works have been performed on the optimization of RRAM based in-memory computing circuits which aims at reducing the number of required steps and RRAMs. Besides BDDs, And-Inverter Graphs (AIGs) have been also used for logic synthesis with resistive memories [11]. However, none of these data structures have been optimized with respect to the cost metrics of in-memory computing circuit design. To the best of our knowledge, optimization for RRAM based design has only been performed on Majority-Inverter Graphs (MIGs) until now [12].

In this paper, BDDs using the MUX design proposed in [9] are optimized with respect to the number of RRAMs and computational steps which are related to the area and delay of the resulting circuits, respectively. The proposed multi-objective genetic algorithm minimizes both mentioned criteria for parallel evaluation where one MUX is considered for each node in a BDD level to synthesize the given function.

The remainder of this paper is organized as follows. Section II contains the basic principles of RRAM based logic and a brief description of the existing BDD optimization approaches. The proposed BDD optimization algorithm is discussed in Section III. Section IV presents the experimental results and Section V concludes the paper.

II. BACKGROUND

This section describes the employed RRAM based 2-to-1 multiplexer circuit as well as discussing the state-of-the-art BDD optimization approaches. In order to keep the paper self-contained the implementation of the basic resistive IMP
which initial values are replaced with intermediate results or the

The corresponding implication steps of the MUX realization

The implementation requires six computational steps and five

unchanged during operations and are called input RRAMs [9].

The FALSE operation can be executed by applying a

positive voltage \( V_{\text{COND}} \) has a magnitude smaller

in order to initialize their desired initial states. The remaining steps are

performed by sequential IMP operations that are executed by

applying simultaneous voltage pulses \( V_{\text{COND}} \) and \( V_{\text{SET}} \).

B. BDD optimization methods

BDDs are a representation for Boolean functions that are

canonical for a fixed variable ordering. BDD optimization is

the task of finding a variable ordering which minimizes the

considered cost metrics, e.g., the number of nodes in the BDD. Improving the variable ordering to find the optimum BDD is

NP-complete [13].

BDD optimization is mainly defined as minimization of the

size of diagram, that is the number of BDD nodes. Exact

methods are the only category of classical size-driven BDD

optimization approaches that guarantee to determine the optimal

variable ordering. Nonetheless, the high order of run-time

and complexity is a serious drawback of these methods [1].

Sifting [14] is a well-known dynamic reordering based BDD

minimization technique that aims at finding the best position

of each input variable, assuming that the relative order of

other variables does not change. For this purpose, the adjacent

variables are swapped and the size of the resulting BDDs are

recorded. Finally, the optimum BDD is characterized by the

the variables fixed at positions which led to the BDD with

the minimum number of nodes. Heuristic approaches such as

Simulated Annealing (SA) [15] and Genetic Algorithms (GAs)

[16] have shown better performance than sifting. SA starts with

a randomly generated variable ordering. In each iteration, the

current ordering is kept or replaced by a neighboring ordering

based on a transition probability which allows uphill moves

in order to escape a local minimum. In [16], GA is combined

with sifting such that the initial population is first optimized by

sifting. GA terminates if the optimal variable ordering remains

unchanged after a certain number of generations. In the last step,

the optimal ordering is sifted to further improve the resulting

BDD’s size if possible.

The number of paths has been also considered as an

objective for BDD optimization due to its importance in some

applications. Modified Sifting (MS) [17] is a path minimization

method that is structurally similar to sifting with a different

objective. An evolutionary algorithm has been also proposed in

[18] for one-path optimization that has experimentally shown to

be able to achieve higher degree of minimization in comparison

with MS. In [19], a multi-objective evolutionary algorithm for

BDD optimization was proposed to minimize the number of

nodes and paths simultaneously. The algorithm has shown a

good trade-off between both objectives without any loss of

quality compared to the existing node or path minimization

techniques.

A. RRAM based MUX

As mentioned before, the BDD nodes can be directly

mapped to MUXes using resistive switches. The basic principles of

resistive logic and the structure of an RRAM based MUX are

explained in the following.

Any Boolean function can be expressed in one of the

standard forms by using only Material Implication (IMP) and

FALSE operation that always assigns the logic value 0. Fig. 1

demonstrates the basic implementation of an IMP gate including
	two resistive switches denoted by \( P \) and \( Q \) that are connected by
	a common horizontal nanowire to a load resistor \( R_{G} \). Tri-state voltage drivers with a high-impedance output state for
	the undriven case control the vertical nanowires. Three voltage levels \( V_{\text{SET}}, V_{\text{COND}} \) and \( V_{\text{CLEAR}} \) are applied to perform IMP

and FALSE operations by placing the switches in low-resistance state (logic 1) or high-resistance state (logic 0).

The FALSE operation can be executed by applying a

positive voltage \( V_{\text{CLEAR}} \) to a switch. A switch can be assigned

to logic 1 by applying \( V_{\text{SET}} \), i.e., a negative voltage larger than a threshold required to change the state of the driven

device, to its voltage driver. \( V_{\text{COND}} \) has a magnitude smaller than \( V_{\text{SET}} \) which cannot cause any state change. Nonetheless,

applying together \( V_{\text{SET}} \) to \( Q \) and \( V_{\text{COND}} \) to \( P \) simultaneously, the IMP operation can be implemented by interaction of two

pulses through \( P, Q \), and the load resistance. The conditional switching condition depends on the current logical states of \( p \)

and \( q \), such that switch \( Q \) is set to 1 if \( P \) is in high-resistance \((p = 0) \) while it remains unchanged when \( P \) is in low-resistance \((p = 1) \) [7].

Fig. 2 shows the RRAM based MUX proposed in [9]. The

implementation requires six computational steps and five

RRAMs of which three, named \( S, X \) and \( Y \), are work RRAMs

which initial values are replaced with intermediate results or the

final output. The two other resistive switches, \( A \) and \( B \), remain

unchanged during operations and are called input RRAMs [9].

The corresponding implication steps of the MUX realization

shown in Fig. 2 are as follows:

Step 1: \( S = s, A = a, B = b, X = 0, Y = 0 \)
Step 2: \( x \leftarrow s \) IMP \( x = \bar{s} \)
Step 3: \( x \leftarrow b \) IMP \( x = \bar{b} + \bar{s} \)
Step 4: \( y \leftarrow x \) IMP \( y = b \cdot s \)
Step 5: \( s \leftarrow a \) IMP \( s = \bar{a} + s \)
Step 6: \( y \leftarrow s \) IMP \( y = a \cdot \bar{s} + b \cdot s \)

In the first step, a \( V_{\text{CLEAR}} \) is applied to resistive switches \( X \) and \( Y \) to execute the FALSE operation. At the same time, the

other three switches are also prepared by applying appropriate

pulses, \( V_{\text{SET}} \) or \( V_{\text{CLEAR}} \), to their voltage drivers in order to

initialize their desired initial states. The remaining steps are

performed by sequential IMP operations that are executed by

applying simultaneous voltage pulses \( V_{\text{COND}} \) and \( V_{\text{SET}} \).

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{fig1.png}
\caption{IMP operation. (a) Implementation of IMP using resistive switches. (b) Truth table for IMP \((q' \leftarrow p \text{ IMP } q)\) [7]}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{fig2.png}
\caption{RRAM based MUX circuit [9]}
\end{figure}
III. BDD optimization for RRAM based design

Optimization of RRAM based BDDs in this work is carried out as a bi-objective problem aiming at minimizing the number of RRAMs and computational steps simultaneously, i.e., finding a trade-off between area and delay of the resulting circuits. For this purpose, we have exploited a non-dominated sorting based genetic algorithm that has been experimentally proven useful in multi-objective BDD optimization [19]. In this section, we first present how the objective functions, i.e., the number of required RRAMs and steps, are calculated. Then, we explain the framework of our algorithm and the employed variation operators.

A. Objective functions

In [9], BDDs were evaluated for two types of serial and parallel implementations for RRAM based design. Serial implementation requires only one RRAM based MUX which can be reused later by other BDD nodes. That means one MUX in addition to a number of RRAMs required for fanouts and complemented edges [20] suffice to build the BDD. Nonetheless, in the presence of complemented edges, the number of required steps to evaluate the Boolean function is greater than six times the number of BDD nodes. It was experimentally shown in [9] that the number of computational steps can rapidly increase for functions with a large number of inputs. In order to escape heavy delay penalties, BDDs are evaluated in parallel in this work.

In the parallel implementation, each time one BDD level is evaluated entirely starting from the level designating the last ordered variable to the first ordered variable the so-called root node. Therefore, regardless of the possible fanouts and complemented edges in the BDD, the number of required RRAMs is five times the maximum number of nodes in any BDD level. Although, the number of RRAMs is increased in comparison with the serial evaluation, the number of computational steps can be remarkably lowered to the number of BDD levels, i.e., the number of input variables of the given function.

Table I shows how the objective functions are evaluated for optimization. However, the larger part of the objective functions are explained above, some additional RRAMs addressing complemented edges and fanouts are still required. Every complemented edge in the BDD requires a NOT gate to invert its logic value. As shown in step 2 discussed in the previous section, inverting a variable can be executed after an IMP operation with a zero loaded RRAM. Accordingly, for each MUX with a complemented input an extra RRAM should be considered and set to FALSE \((Z = 0)\) that can be performed in parallel with the first loading step without any increase in the number of steps. Then, an IMP operation should be executed to complete the logic NOT operation. It is obvious that the required IMP operations for all complemented edges in a level can be carried out simultaneously that means for any level with ingoing complemented edges only one extra step is required. This implies that the number of additional steps required for inverting all of the complemented edges cannot exceed the number of BDD levels. Therefore, the number of steps to evaluate a BDD possessing complemented edges is equal to the number of BDD levels with ingoing complemented edges besides the basic value of level counts.

It is obvious that the RRAMs keeping the outputs of each BDD level can be assigned to the inputs of the next successive level and be reused without any loss of information. Nonetheless, the results of nodes targeting levels which are not right after their origin level might be lost during computations if their corresponding RRAMs are rewritten by the next operations. Thus, we consider extra RRAMs for such nonconsecutive fanouts to retain the result of their origin nodes to be used as an input signal of their target nodes. The required number of RRAMs for this is equal to the maximum number of such fanouts over all BDD levels. This will not increase the number of steps because copying the results of nodes with nonconsecutive fanouts in additional RRAMs and using the stored value in the fanouts’ targets can be performed simultaneously in the first data loading step of nodes on the both sides of the fanouts.

B. Genetic algorithm framework

Here, we explain the framework of our multi-objective BDD optimization algorithm. Genetic and more generally evolutionary algorithms are known as powerful search tools and are of high interest for multi-objective optimization especially for solving NP-complete problems such as BDD optimization. The employed BDD optimization algorithm is based on NSGA-II (Non-dominated Sorting Genetic Algorithm) [21] which has shown excellent performance for the optimization problems with two or three objectives.

In general, an evolutionary algorithm consists of iteratively applying two processes of selection and variation to a population of possible solutions, i.e., a subset of the search space. Selection decides which individuals in the population are good enough to be used as parents by variation operators for generating an offspring, as well as determining the survival strategy for filling the population of the next generation. In our genetic algorithm, we have utilized non-dominated sorting relation to discriminate solutions during selection. Based on Pareto-dominance, an individual \(x\) is said to dominate \(y\) if none of its objective functions are greater than the corresponding objective function in \(y\), and \(x\) at least has one objective function smaller than the corresponding one in \(y\). According to non-dominated sorting, every individual should be assigned to a fitness value representing its overall level of dominance in such a way that the first level of dominance contains non-dominated solutions which are not dominated by any other solution in the population.

The general framework of our Multi-Objective Genetic Algorithm (MOGA) is described in Algorithm 1. First, a population of size \(N\) consisting of variable orderings representing a set of BDDs is generated randomly. Then, the initial population is evaluated to assign the corresponding number of RRAMs and steps to each BDD. In step 3, the population is classified into non-dominating fronts shown by \(\{F_1, F_2, \ldots\}\) such that members of each front are incomparable based on dominance. Steps 6–18 are iterated for a certain number of generations.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(N)</td>
<td>No. of input variables</td>
<td>Given</td>
</tr>
<tr>
<td>(FO)</td>
<td>Maximum no. of nonconsecutive fanouts in any BDD level</td>
<td>Given</td>
</tr>
<tr>
<td>(CE_i)</td>
<td>No. of ingoing complemented edges in the (i^{th}) BDD level</td>
<td>Given</td>
</tr>
<tr>
<td>(N_{Li})</td>
<td>No. of nodes in the (i^{th}) BDD level</td>
<td>Given</td>
</tr>
<tr>
<td>(L)</td>
<td>No. of BDD levels with ingoing complemented edges</td>
<td>Given</td>
</tr>
<tr>
<td>(R)</td>
<td>No. of RRAMs</td>
<td>(\max {5 \cdot N_{Li} + CE_i + FO})</td>
</tr>
<tr>
<td>(S)</td>
<td>No. of computational steps</td>
<td>(6 \cdot N + L)</td>
</tr>
</tbody>
</table>
The chosen parents for reproduction are results of a binary tournament selection, that selects the individual with lower fitness between two randomly chosen individuals. In step 6, an offspring $Q_i$ with the same size as the current population $P_i$ is created after applying evolutionary operators such as recombination and mutation to the selected parents. Thereafter, $Q_i$ is evaluated and the union of the children and the current populations, $R_i$ is sorted. Starting from the first non-dominating front, individuals are copied into the population of the next generation $P_{i+1}$. This procedure is continued until the size of the front is greater than the remaining slots in $P_{i+1}$. In this case, the front’s members are sorted based on their density information and then $P_{i+1}$ is filled with the first ordered lesser crowded individuals.

**Algorithm 1: MOGA for BDD optimization**

```
1: $P_0 \leftarrow$ InitializePopulation
2: EvaluateObjectiveFunctions($P_0$)
3: $\{F_1, F_2, \ldots\} \leftarrow$ Non-dominatedSort($P_0$)
4: $t \leftarrow 0$
5: while the stopping criterion is not met do
6: $Q_t \leftarrow$ MakeOffspringPopulation($P_{i+1}$)
7: EvaluateObjectiveFunctions($Q_t$)
8: $R_t \leftarrow P_t \cup Q_t$
9: $\{F_t, F_2, \ldots\} \leftarrow$ Non-dominatedSort($R_t$)
10: $P_{t+1} \leftarrow \emptyset$
11: $i \leftarrow 1$
12: while $|P_{t+1}| + |F_t| \leq N$ do
13: $P_{t+1} \leftarrow P_{t+1} \cup F_t$
14: $i \leftarrow i + 1$
15: end while
16: DistanceSort($F_t$)
17: $P_{t+1} \leftarrow P_{t+1} \cup F_t[1 : (N - |P_{t+1}|)]$
18: $t \leftarrow t + 1$
19: end while
```

It might be desired to design smaller circuits at a fair cost of delay or vice versa. In [19], relation priority dominance was used in multi-objective BDD optimization to allow preference to the more significant objectives. Our genetic algorithm is enhanced with a priority vector which can be set by the user to perform optimization with preference to the number of RRAMs or steps.

**C. Operators**

In this section, the employed variation operators for creating the offspring population are briefly explained. We have utilized two recombination operators including Partially Matched Crossover (PMX) [22] and inversion which maintain validity of the variable permutations. PMX breaks selected parents into three sections by choosing two random positions. Then, two children are created by combining the sections from both parents in such a way that no previously used index is repeated inside the variable ordering. Applying inversion, a single parent is divided into three sections similarly to PMX. Finally, the order of variable indices in each section is inverted to produce a child.

The created children are also slightly changed by use of three mutation operators which are applied based on given probabilities. One mutation operator exchanges the contents of two randomly selected variable indices. The second mutation scheme performs the previous operator on the same child for two times. In the third employed mutation operator a random position in the child is selected and its value is exchanged for an adjacent index.

**D. Example**

Fig. 3 shows an example with two BDDs both representing a 4-variable 2-output Boolean function. The left BDD has the initial ordering, whereas the second BDD has the ordering obtained by MOGA. The number of required RRAMs for computing BDD levels (5 · NL + CE) is equal before and after optimization since both BDDs have a maximum number of two nodes and one ingoing complemented edge over all levels. However, there is a nonconsecutive fanout of node $x_3$ targeting $x_1$ before optimization requiring an extra RRAM to maintain the intermediate result. In the optimized BDD the inputs of all of the nodes come from the consecutive levels or the constant 1 which has reduced the number of required RRAMs by 1. The number of computational steps has been also reduced after optimization since one level has been released from complemented edges.

As can be seen, the numbers of RRAMs and steps decrease although the number of BDD nodes increases. The effect of BDD optimization sounds to be too small for the example function by reducing each one of the cost metrics only by one. Nevertheless, this reduction can be much more visible for larger functions due to the higher possibility of finding BDDs with smaller number of nonconsecutive fanouts, complemented edges and level sizes caused by larger search space.

**IV. EXPERIMENTAL RESULTS**

The results of experiments on 25 benchmark functions taken from LGSynth91 [24] are presented in this section. The number of input variables of the selected functions are in range from 7 to 135. We have used the CUDD package [23] for BDD representation and assessment of the optimization results. For each benchmark function, MOGA has been run 10 times with a termination criterion of 500 generations. The population is three times as large as the number of inputs of each function with a maximum allowed size of 120. The probabilities for PMX and inversion are set to 0.98 and 0.01 respectively. The mutation probability is distributed identically over the three operators with a value of $1/n$, where $n$ is the number of input variables.

Table II compares the results of MOGA with the BDDs generated by the initial variable orderings given by CUDD.
Values given in Table II are the best found trade-off between objective functions chosen manually from the final populations of all 10 runs. The results in Table II show that BDDs found by MOGA require smaller number of RRAMs in comparison with the initially ordered BDDs. More precisely, the sum of RRAM counts by CUDD for the whole benchmark set is more than 58 times the corresponding amount of the optimized BDDs. The sum of the number of required steps to evaluate BDDs is also lowered by 4.16%. It should be noted that in parallel implementation the number of steps is close to the lowest possible value and does not depend on the BDD characteristics too much. As shown in Table I, the only cost metric that can be reduced by optimization is the number of levels with complemented edges. Therefore, the number of steps are not highly affected by optimization. A similar situation occurs in BDD optimization for serial implementation where the number of required RRAMs can be quite constant for a given function. Nonetheless, the optimization results show a considerable reduction in the values of objective functions.

MOGA is able to handle objective priorities when a fair increase in the area or delay can be tolerated for achieving higher minimization with respect to one criterion. Results of optimization with priority to the number of RRAMs and computational steps are demonstrated in Table III. As shown in the Table III, MOGA with priority to the number of RRAMs has obtained the smallest sum of RRAMs and as expected the smallest number of steps are provided by setting higher priority to steps. In both cases, a decrease in the objective function of higher importance has led to an increase in the value of the other objective. As discussed before, the number of required steps for evaluation cannot be dramatically lowered by optimization. This explains why MOGA with priority to the number of steps has resulted in higher overhead for a smaller reduction in $S$ while a greater reduction in $R$ has not increased the number of steps considerably.

In Table IV, our optimization results are compared with results given in [9]. The number of computational steps in results by Chakraborti et al. [9] are calculated by using a different function from ours given in Table I. In [9], the maximum number of complemented edges in any BDD level is added to the basic value relating to the number of input variables of the Boolean function. As explained before, we have considered one extra RRAM for any complemented edge in the diagram. Thus, for any level containing complemented edges one extra step is sufficient to execute all the required IMP operations simultaneously.
A comparison of Table IV and Table II reveals that the results given in [9] do not represent the initially ordered BDDs. Actually, a kind of random optimization is done which tries a number of randomly created variable orderings and keeps the best found BDDs for each benchmark function. MOGA has achieved better performance in both objective functions. Regardless the difference in the number of steps which is slightly affected by the different equations for $S$, MOGA has also decreased the total number of required RRAMs. The sums of the number of RRAMs and steps required to evaluate BDDs of all benchmark function show a reduction of 25.28% and 31.74%, respectively.

V. CONCLUSION

RRAM based design has gained high interest for its possible applications in various domains. In logic circuit design using resistive switches, it is aimed to reduce the required number of RRAMs and computational steps. Especially the latter one can be more costly for larger circuits while using higher number of RRAMs does not necessarily cause high area overhead due to their tiny dimensions. The presented approach employs BDDs for RRAM based logic circuit design and performs multi-objective optimization by a genetic algorithm in order to attain smaller and faster circuits. The proposed genetic algorithm is capable of considering user preference to any of the objectives that might be of interest in some applications. Performance evaluation and comparison of experimental results reveal that our genetic algorithm fairly lowers both criteria and finds a good trade-off between them.

ACKNOWLEDGMENT

This research was supported by the German Research Foundation (DFG) (DR 287/23-1) within a Reinhart Koselleck project, the University of Bremen’s graduate school SyDe, funded by the German Excellence Initiative, and H2020-ERC-2014-ADG 669354 CyberCare.