BDD Ordering: A Method to Minimize BDD Size by Using Improved Initial Order

MD. BALAL SIDDIQUI1, MANU BANSAL2
1, 2 Electronics and Communication Engineering Department , Thapar University, Patiala, India
balalisd@gmail.com, mbansal@thapar.edu

ABSTRACT

Binary Decision Diagrams gained widespread use in the field of logic synthesis, verification and testing of Digital circuits. For PLA based design of digital circuits the number of internal nodes of Binary Decision Diagrams describe physical area on chip. The size of the Binary Decision Diagrams i.e. number of internal nodes strongly depends on the order of the variables in Binary Decision Diagrams. Many algorithms had been proposed in past decades to order variables in Binary Decision Diagrams which uses a simple random initial order and then apply different schemes to generate new order using this initial order. In this paper we have introduced a new method which generates an improved initial order which is used in different proposed ordering algorithms as a seed order and new orders are generated using this improved initial order instead of a simple random initial order.

Keywords: BDD Variable ordering, BDD optimization, Ordering algorithm

[1] INTRODUCTION

Binary Decision Diagrams are first proposed by Lee [1] to represent the Boolean functions. Bryant found that the ordering of input variables in Binary Decision Diagrams has strong effect on number of nodes and he proposed Reduced-Ordered Binary Decision Diagrams (often call as BDD) [2]. In last decade BDD has emerged as the most effective style of representation and manipulation of Boolean functions [3]. For Boolean functions having very large number of input variables, BDD use very efficient memory when compare with other style of representation [2]. BDD has found application in the field of synthesis, verification, test generation and fault simulation of digital systems [3, 4].

The BDD is a rooted, directed, acyclic graph having one root node, two terminal nodes and many internal nodes [5]. The number of internal nodes in BDD is known as size of the BDD. For PLA based design of the digital functions the size of the BDD represents chip area. So to reduce area utilization of the function we have to reduce size of the BDDs [6]. The size of a BDD strongly depends on the ordering of input variables. For a BDD have compact size with a good variable order may have exponential increase in size when a bad order is chosen [7]. For example a BDD, which represent the carry-out of an adder, having size n for a good variable ordering may have size $2^{n^2}$ when some other ordering is chosen [8]. Many variable ordering methods has been proposed in last decades. These methods include static and dynamic techniques. Static techniques are applied before constructing the BDD to generate a order depend upon the implemented function. Like the static technique used by Fujita which do a depth-first traversal through a circuit from the output to the inputs [9]. Even though static techniques are good over dynamic techniques, it requires the prior knowledge about function's behavioral, like effect of each input variables on function, which is not known in many cases. Dynamic techniques are focused on generating new variable orders to improve the size of the already constructed BDD. One of the such technique is proposed by Ruddel is sifting algorithm [10]. It is the swapping of two variables. It first select one variable, move it to each position and find the number of nodes.

Then the variable is fixed at the position which gives minimum number of nodes. Now the next variable is picked and moved to each position except the fixed positions by previous variables and then its position is fixed for minimum number of nodes. In this way all variables are sifted from its old position to its new position. The final ordering by fixing all variables is the improved ordering find by sifting algorithm. This sift algorithm decrease the number of permutations of ordering from $n!$ to $n^2$ but in many cases the size of BDD is far from optimal. Many other methods uses genetic algorithm [11-14] and different heuristics [15-16] to generate new ordering. Genetic algorithm based approach used genetic operations like crossover and mutation to generate new variable ordering based on some fitness criteria. Genetic algorithm bases approaches have good run time but when compare to other techniques, result is not quite improving. Simulated annealing based approach is proposed in [17] which gives better result but on the cost of long run time. Variable ordering by scatter search is proposed in [15] which gives better result with better run time in many cases but with limited number of input variables. For circuits having large number of input variables scatter search based method is not satisfactory. Prasad had proposed a new graph based approach for ordering which use different graphical approaches and shows improvement on many previous works [18]. Although many methods are present to minimize the BDD size they still shows exponential variation in BDD size for many functions. In this paper we have proposed a new method which use with other standard method gives improved result.
The organization of this paper is as follows: Section 2 will introduce the BDD and ordering issues in BDD. Section 3 will discuss our new method to generate improved initial order. Experimental results will be included in Section 4. Finally in section 5 conclusion of this paper will be discussed.

[2] BDD AND ORDERING ISSUES

Binary Decision Diagrams are single rooted direct acyclic graph used to describe a switching function. It was first proposed by Lee [1] which was based on Shannon decomposition in which each nodes of the decision diagram represents some sub function. The BDD is later ordered and reduced based on some rules proposed by Bryant. Ordered means BDD in which the ordering of variables is consistent on all paths of the graph [9]. Each non terminal node in BDD is labeled by an input variable and have two outgoing branches known as 0-branch and 1-branch. When both the branch of a node points to same node then the node is eliminated. Also when two nodes have identical branches then one node is merged to another node. These reduction rules are shown in Fig. 1. In figures the dotted line shows 0-branch and other line shows 1-branch. The Reduced-ordered Binary Decision Diagrams are often called as BDD. It gives canonical representation of a Boolean function[2]. The order of the input variables has a strong effect on size on BDD. The size may vary exponentially if a bad order is chosen. In Fig. 2 two BDD for function f = ab+a’c+bc’d with different variable order has shown.

![Fig. 1. Reduction rules for Reduced-Ordered BDDs.](image1)

![Fig. 2. BDDs for function f = ab+a’c+bc’d with different ordering.](image2)

In Fig. 2 BDD with order a,b,c,d, has size of 6 nodes whereas when order a,c,d,b is used the size is reduced to only 4 nodes. The example function shown in Fig. 2 is not complex one and has only four variables. In complex functions when number of variable are large the effect of ordering also vary exponentially from the optimal size. There are many BDD packages available for manipulation of Binary Decision Diagrams. CUDD is developed by Boulder in University of Colorado [19]. BuDDy is another popular package [20] which is developed by Jorn Lind-Nielsen. These packages use standard ordering method like windows method, Sift algorithm, random method etc.. While computing new order from these inbuilt standard methods these packages uses a general initial order. We have proposed a method in which a new improved seed order is generated which use as initial order to find new order by these inbuilt standard methods in BDD packages.

[3] PROPOSED METHOD

The dynamic algorithms proposed so far generally used a simple initial order to calculate new orders. We observed that if a improved initial order is used in many proposed algorithms then the nodes calculated by same algorithm is reduced.

In our method we first checked the impact of each variable by moving it at m different positions where m = $(p*n)/l$, and n is the number of input variables, l is the number of points at which size is calculate and p varies from 0 to l. An example is shown in fig 3 for a function having 20 variables for which we have taken l as five.

![Fig. 3. Swapping Variable at different positions.](image3)

In fig. 3 we have moved first variable (variable 1) at five different positions and calculated the number of nodes N for each five positions, the number of points to calculate may vary with number of variables. The step size i.e. difference between two successive points will increase with increase in number of variables. After calculating size at all the l points the variable is restored at its original position and the minimum size from the list of sizes at l points is stored. The process shown in fig. 3 for first variable is repeated for all the n variables. Now from the list of minimum size of each variable we give new position to the variable corresponding to their minimum size in ascending order i.e. the variable corresponding to lowest minimum size gets first position then the variable corresponding to next minimum size gets second position and so on. The proposed method is shown by a flow chart in fig. 4. The new generated order is now used as initial order for well known standard variable ordering algorithms.

[4] EXPERIMENTAL RESULT

We have applied our method on different benchmark circuits available on LGsynth93 benchmark circuit suits. The method described here is implemented in C language on a Linux machine. For BDD manipulation we have
used BDD package Buddy-2.4 [20]. We have generated the improved initial order and applied it to the standard reordering techniques that have been included in the Buddy-2.4, such as Sifting algorithm and window permutations methods WIN2ite, WIN2, WIN3. The results for different benchmark circuits is shown in table 1.

![Flow Chart of the proposed method](image)

**Fig. 4.** Flow Chart of the proposed method

The columns #i and #o shows numbers of input variables and number of output functions of the benchmark circuits respectively. For each of the four method we have calculated number of nodes with simple initial order and with improved initial order. This is shown in column simple initial order and proposed initial order respectively, we observed that the size has improved in many circuits. Specially for the circuits having large number of input variables like vg2 the variation in size with the order is quite high. For one circuit 5xp1 there is increment size. We have taken result Total as sum of all the circuits and calculated percentage improvement by taking size obtained from all the benchmark circuits together with both the initial order. We observed that the improvement is quite high for sift and Win2ite which is around 11.22 % and 5.43 % respectively. for other techniques the improvement is not so impressive.

<table>
<thead>
<tr>
<th>Benchmark circuits</th>
<th>#i</th>
<th>#o</th>
<th>Sift</th>
<th>Win2ite</th>
<th>Win2</th>
<th>Win3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Simple initial Order</td>
<td>Proposed initial order</td>
<td>Simple initial Order</td>
<td>Proposed initial order</td>
</tr>
<tr>
<td>Cordic</td>
<td>23</td>
<td>2</td>
<td>39</td>
<td>48</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>b12</td>
<td>15</td>
<td>9</td>
<td>65</td>
<td>61</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>5xp1</td>
<td>7</td>
<td>10</td>
<td>78</td>
<td>88</td>
<td>89</td>
<td>89</td>
</tr>
<tr>
<td>clip</td>
<td>9</td>
<td>5</td>
<td>105</td>
<td>94</td>
<td>93</td>
<td>93</td>
</tr>
<tr>
<td>vg2</td>
<td>25</td>
<td>8</td>
<td>286</td>
<td>325</td>
<td>295</td>
<td>295</td>
</tr>
<tr>
<td>sao2</td>
<td>10</td>
<td>4</td>
<td>93</td>
<td>95</td>
<td>88</td>
<td>90</td>
</tr>
<tr>
<td>rd84</td>
<td>8</td>
<td>4</td>
<td>59</td>
<td>59</td>
<td>59</td>
<td>59</td>
</tr>
<tr>
<td>rd73</td>
<td>7</td>
<td>3</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td>43</td>
</tr>
<tr>
<td>misex3c</td>
<td>14</td>
<td>14</td>
<td>467</td>
<td>455</td>
<td>452</td>
<td>451</td>
</tr>
<tr>
<td>misex2</td>
<td>25</td>
<td>18</td>
<td>84</td>
<td>107</td>
<td>81</td>
<td>82</td>
</tr>
<tr>
<td>Total</td>
<td>1319</td>
<td>1171</td>
<td>1375</td>
<td>1300</td>
<td>1301</td>
<td>1299</td>
</tr>
<tr>
<td>% Improvement</td>
<td>11.22</td>
<td>5.43</td>
<td>0.15</td>
<td>0.69</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Comparison of results from new proposed initial order with simple initial order for different algorithms
[5] CONCLUSIONS
The new scheme to use an improved initial order is an improvement over many of the previous methods which use a simple initial order. The method is effective for sifting algorithms but this can also be applied to other heuristics like genetic algorithms where we need to generate good initial population. In our proposed method the node count is improved for many circuits which implies improvement in chip area but there are other concerns like timing and switching activity which is not verified here. For future work while generating the initial order other concerns like timing and switching activity has to be check and optimize.

REFERENCES