Minimization of the Expected Path Length in BDDs Based on Local Changes*

Rüdiger Ebendt[1] Wolfgang Günther[2] Rolf Drechsler[1]

[1]Institute of Computer Science
University of Bremen
28359 Bremen, Germany
{ebendt,drechsle}@informatik.uni-bremen.de

[2]CL DAT TDM VM
Infineon Technologies
81730 Munich, Germany
wolfgang.guenther@infineon.com

Abstract— In many verification tools methods for functional simulation based on reduced ordered Binary Decision Diagrams (BDDs) are used. The evaluation time for a BDD can be crucial and is measured by the expected path length of the BDD.

In this paper a new technique for BDD minimization with respect to the expected path length is suggested to reduce evaluation time. It is based on sifting and, unlike previous approaches, performs variable swaps with the same time complexity as the original sifting algorithm.

Another field of application for BDDs is logic synthesis, often targeting Pass Transistor Logic (PTL) because of low power and low cost. A minimization of BDD size and chip area can lead to poor timing performances. We suggest to also use our method here, as the resulting BDDs show a very low maximal and average path delay. This supports the synthesis of high-speed PTL circuits at low area overhead.

Experimental results are given to show the efficiency of our approach.

I. Introduction

Reduced ordered *Binary Decision Diagrams* (BDDs) are a data structure for efficient representation and manipulation of Boolean functions. They were introduced in [5] and are frequently used in formal verification and logic synthesis.

In many verification tools methods for functional simulation based on BDDs are used. A crucial point here is the time needed to evaluate a function using a representing BDD.

Boolean functions $f\colon \{0,1\}^n \to \{0,1\}^m$ can be viewed as a Boolean relation representable by its *Characteristic Function* (CF). Using a BDD for the CF of this relation, the evaluation time is $O(m+n)$ [1, 21]. The time complexity of function evaluation using shared BDDs directly representing the function f is higher, $O(m \cdot n)$ [1, 21]. However, in functional simulation based on BDDs, shared BDDs directly representing the function f often have to be used instead of BDDs for CFs, since the sizes of BDDs for CFs tend to be too large to preserve practicality [10]. For this reason, there is a strong demand for algorithms speeding up evaluation time for shared BDDs.

Another field of application for BDDs is logic synthesis. *Pass Transistor Logic* (PTL) is often targeted because of low power consumption, low cost and the possibility to consider layout aspects during the synthesis step, e.g. see [17, 16].

Classically, BDD optimization was done with respect to the number of nodes, i.e. BDD size. This reduces the memory and runtime needed for the representation and manipulation of Boolean functions. In the case of PTL synthesis, minimization of BDD size directly transfers to a smaller chip area, but it can lead to chips with poor timing performance [14]. Recently approaches based on Rudell's sifting algorithm have been proposed, which optimize BDDs targeting the delay of resulting circuits [14, 20]. But these techniques either fail to achieve strictly local operations during variable swaps [14], resulting in high run times, or they apply simplified cost functions [20] and thus can produce results which are far away from the true optimization objective.

In this paper we describe a new fast technique to optimize BDDs with respect to the expected path length (EPL), i.e. the average number of variable tests needed to evaluate an assignment of the inputs. It is based on sifting and, unlike previous approaches, performs variable swaps with the same time complexity as the original sifting algorithm[1]. Our method is called EPL sifting. The BDDs yielded support fast functional evaluation for functional simulation.

We suggest EPL sifting also for path delay minimization in BDD-based PTL synthesis, since the metric EPL also models the average gate delay in PTL networks.

Experiments show that with EPL sifting reductions in the length of critical paths up to 48.9% can be observed on benchmark functions. Moreover, our method is faster than the approach oriented towards the maximum path length by up to two orders of magnitude.

II. Preliminaries

Boolean variables (denoted by Latin letters) are bound to values in $\mathbf{B} := \{0,1\}$. It is well-known that a Boolean function $f\colon \mathbf{B}^n \to \mathbf{B}$ over the variable set X_n can be represented by a *Binary Decision Diagram* (BDD) [5], i.e. a directed acyclic graph where a Shannon decomposition

$$f = x_i f_{x_i=1} + \overline{x}_i f_{x_i=0} \quad (1 \le i \le n)$$

is carried out in each node. In the following, only reduced, ordered BDDs are considered and for briefness these graphs are called BDDs. Redundant nodes are assumed to be eliminated and variables are encountered at most once and in the same order (the "variable ordering") on every path from the root to a terminal node. For more details see [5].

Formally, this fixed ordering can be expressed with a mapping

$$\pi\colon \{1,\ldots,n\} \to \{x_1,\ldots,x_n\}$$

of each BDD level to a variable. Thus we write $x_i = \pi(k)$, if variable x_i is the k-th element of the variable ordering, i.e. x_i is in the k-th level of the BDD.

BDDs are defined analogously for multi-output functions $f\colon \mathbf{B}^n \to \mathbf{B}^m$, using a graph for each of the m single-output functions for the *shared* BDD representation. In the following we assume shared BDDs with *Complement Edges* (CEs) [4] without mentioning it further. (Note that all results reported

*This work was supported in part by DFG grant DR 287/8-1.

[1]Recently, the minimization of the expected path length with a sifting approach independently has been studied in [18].

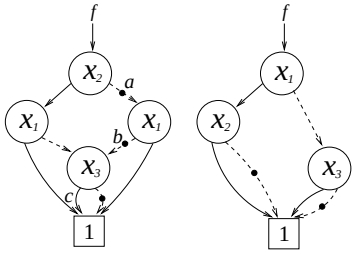

Fig. 1. Two BDDs for f: $(x_1, x_2, x_3) \mapsto x_1 \cdot x_2 + \overline{x_1} \cdot x_3$.

here directly transfer to BDDs without CEs.) A BDD with CEs has exactly one terminal node which we denote **1**. Each other node v has two successors, one is reached via a 1-edge and is denoted *then*(v), the other one is reached via a 0-edge and is denoted *else*(v). We call *then*(v) the 1-son and *else*(v) the 0-son of v. A node v in a BDD is labeled with the variable tested at the node, denoted var(v). Evaluation of an assignment $b = (b_1, \ldots, b_n) \in \mathbf{B}^n$ starts at one of the output nodes and traverses the path along the edges chosen according to the values assigned to the variables by b.

Paths are denoted as alternating sequence of nodes v_i and edges e_i, i.e. $(v_1, e_1, \ldots, e_{k-1}, v_k)$. The length of a path p is the number of non-terminal nodes occurring on p, denoted $\lambda(p)$.

For an example of an evaluation and the traversed path see Example 1.

Example 1 Consider the left BDD given in Fig. 1. The evaluation for $b = (0,0,1)$ starts at the output node for f, which is the root node of the BDD. Assignment b assigns x_2 to 0, x_1 to 0 and x_3 to 1. According to these values, the path along the corresponding edges labeled a, b and c is chosen (1-edges are depicted with solid lines, 0-edges with dashed lines). As an even number of CEs (indicated by a black dot on the according edge) is encountered on this path finally reaching the terminal node labeled 1, the function value is 1 (otherwise, i.e. if we had encountered an odd number of CEs, the function value would be 0). In fact we have $f(0,0,1) = 1$.

Note that the path along a, b and c is of maximal length three, whereas the right BDD has a maximal path length (MPL) of only two. This is because the right BDD has been optimized, which can be achieved by dynamic reordering, i.e. sifting (*end of example).*

Sometimes we may choose to assign values to only the first few variables in the ordering, thus considering a possibly shorter prefix $a = (b_1, \ldots, b_k)$ $(k \leq n)$ of a (full) evaluation b. Evaluation of a stops at a possibly non-terminal node v, representing the cofactor $f_{x_1=b_1,\ldots,x_k=b_k}$, for which we also write f_a.

III. Previous Work

In this section we review the BDD metric EPL, which was suggested in [15] as measure for evaluation time in BDD-based functional simulation. It has also been used recently in a comparison of different methods to evaluate multi-output logic functions with BDDs in [10]. EPL expresses the expected number of variable tests needed to evaluate an input assignment along a path from an output node to **1**.

Let F be a BDD and let p_i be the i-th path in an enumeration of all paths from output nodes to the terminal node in F. The number of all these paths is denoted P. Let $\mathrm{pr}(p_i)$ be the probability of an evaluation traversing path p_i. Then for the EPL of F, denoted $\varepsilon(F)$, we have

$$\varepsilon(F) = \sum_{i=1}^{P} \lambda(p_i) \cdot \mathrm{pr}(p_i). \tag{1}$$

A path p is weighted in this formula with the probability of being chosen during evaluation. Minimizing $\varepsilon(F)$ means shortening the path lengths with a high probability, thus minimizing the expected path length EPL or, in other words, the average evaluation time.

Eq. (1) is not suitable for an efficient algorithm to compute EPL, as P can grow exponentially in n, even if the size of the BDD only grows linear in n [9]. In [15], the authors give a formula useful for computing EPL in time proportional to the BDD size: the following equation expresses the expected number of variable tests for an evaluation starting from a node v and ending at **1** (this quantity denoted $\varepsilon(v)$). Thereby we denote the probability that a variable x is assigned to a value $b \in \mathbf{B}$ with $\mathrm{pr}(x = b)$.

$$\varepsilon(v) = \begin{cases} 0, & & & v = \mathbf{1} \\ 1 & + & \mathrm{pr}\big(\mathrm{var}(v) = 1\big) \cdot \varepsilon(\mathit{then}(v)), & \text{else} \\ & + & \mathrm{pr}\big(\mathrm{var}(v) = 0\big) \cdot \varepsilon(\mathit{else}(v)) & \end{cases} \tag{2}$$

This formula simply states that $\varepsilon(v)$ is zero, if v already is the terminal node. Otherwise, evaluations starting from v to **1** are either via the then-child or the else-child of v. Hence, $\varepsilon(v)$ is built by a) summing up the resp. ε-values of the child nodes weighted with the probability of the resp. child node being chosen and b) adding one, since the expected length of all paths starting at v must be one larger than that of the child nodes of v: this is due to the additional variable test at v.

In [15] an algorithm to minimize EPL for a given shared BDD was suggested.

The minimization is performed with a window permutation algorithm [11]. After each swap of variables, establishing a new variable ordering, the resulting expected path length of the restructured BDD is determined. Recalculations need to be done in the restructured part of the BDD covered by the window. Therefore, the window (and with that, also the BDD) is split into an upper and a lower part. Each part is updated with one of two different methods. Afterwards, the proposed method to compute the new expected path length needs to determine all direct references from nodes in the upper BDD part to nodes in the lower BDD part. This operation requires touching large parts of the BDD including many levels [6, 8]. The authors also describe a simplified approach, which computes an estimation of the expected path length, computing only the edges between two adjacent levels. However, this means that only a simplified delay model is used.

IV. Applying the Expected Path Length in Logic Synthesis

The BDD characteristic EPL so far has only been used in BDD-based functional simulation (see Section III). Next we give reasons why EPL can also be used for path delay minimization in BDD-based PTL synthesis. We start by clarifying that this metric also models the average gate delay in PTL networks.

EPL expresses the expected number of variables which are tested on a path in a BDD. Variables are tested during evaluation of an input assignment in $\mathbf{B}^n$. Paths in the BDD correspond to chains of pass transistors in a PTL network. EPL then corresponds to the expected number of multiplexors along such a chain which have to pass a signal to determine the resulting output signals of the circuit.

We assume a *unit delay model* [7], i.e. every gate (which is corresponding to a BDD node) is assigned a uniform delay of one. This model is independent of the technology used and has been applied to the unmapped netlists in the synthesis approach of [20]. Then, the EPL models average gate delay for PTL networks derived from BDDs: EPL accounts for the fact

```
(1)  eps_on_level(BDD F, int level) {
(2)      for all nodes v at level level do {
(3)          ε(v) := 1 + 1/2 (ε(then(v)) + ε(else(v)));
(4)      }
(5)  }

(6)  eps_above_level(BDD F, int level) {
(7)      for i := level to 1 do {
(8)          eps_on_level(F, i);
(9)      }
(10) }
```

Fig. 2. Iteratively computing $\varepsilon(F)$.

that the delay occurring on one multiplexor chain may have a larger influence on the average delay than that of a second chain. This is the case if more input signals in the environment of the circuit are passed along the first chain in (statistical) average.

Circuit speed is measured by the delay on a critical path. Here, this path corresponds to the maximum path length (MPL) in BDDs. Examples like the one given in Fig. 1 illustrate the intuition, that BDDs with a small overall path length also show a small length of the longest path: the right BDD has a minimal MPL as well as a minimal EPL, due to a simpler graph structure consisting of less nodes and edges than the left BDD. In Section VI we describe experiments, showing that the MPL of the BDDs yielded by our EPL minimization approach is almost the same as for BDDs directly minimized towards a smallest MPL. Although this meets general intuition, the high consistency of the results is remarkable. Hence, while being a fast method, the proposed technique preserves quality of the results. This significantly contrasts to all previously suggested approaches.

Moreover, the resulting BDDs are also minimized in average gate delay and hence are a good starting point for additional critical path analysis [12, 3, 14]: it is well-known that these techniques can further improve the results by subsequent local reorganization, up to optimizing the delay on a critical path to not exceed the average delay at all [2].

V. A Fast Approach to Minimize the Expected Path Length in BDDs

In this section we describe a new fast method called EPL sifting to minimize the EPL in shared BDDs. It is based on Rudell's sifting [19] and performs variable swaps with the same time complexity as the original sifting algorithm. This contrasts to the approach in [15], which uses a window permutation approach. This algorithm is either forced to touch larger parts of the BDD with every variable swap or to use a simplified estimation of the true cost function EPL.

The main result of this section will be the desired *locality* of the variable swap operation in the new method: only the nodes in the adjacent levels affected by a variable swap have to be processed during a swap (due to the recalculation of values, which have become invalid). It is this locality which makes our approach fast. In general, it is difficult to obtain locality, e.g. for a sifting modification targeting MPL (MPL sifting), to the best of our knowledge, a local approach is not known. Later in Section VI a comparison of the run times of our method and MPL sifting is given.

A. A First Simple Approach

In Fig. 2, a first simple algorithm to compute the ε-values for all nodes in a BDD F is given: for a call **eps_above_level**(F, n) the algorithm proceeds bottom up starting an iteration at the lowest level n onto the highest level 1. On every level the ε-values of nodes at the lower levels, which already have been computed, are used to compute the current values[2].

The characteristic EPL for a shared BDD F representing a Boolean multi-output function $f = (f_i)_{1\le i\le m}$ can be computed by use of the ε-values of the output nodes representing the m single-output functions:

$$\varepsilon(F) = \frac{1}{m}\sum_{i=1}^{m}\varepsilon(o_i), \tag{3}$$

where o_i is the output node representing f_i. Note, that an output node o_i might be used by multiple, functionally equivalent single-output functions, as circuits sometimes repeat an output signal several times.

B. Keeping Track of Local Changes

A change in the ε-value of a node in the uppermost of the levels involved in a variable swap (i.e. a "local" change) influences the ε-value of many nodes above, among them at least one output node. Hence, these values become invalid. Instead of recalculating the values of all these nodes above we choose a much less time-consuming strategy explained in this section: we directly transfer the local changes to changes in the ε-value of the output-nodes, thus correctly updating the global EPL-value. This is possible *without* recalculations in the levels above the swapped levels.

For the following, we always assume a shared BDD F representing a Boolean multi-output function $f\colon \mathbf{B}^n \to \mathbf{B}^m$; $f = (f_i)_{1\le i\le m}$ without further mentioning. We also introduce a new notation: for an edge e on a path in a BDD, we denote the type of the edge with $\mathrm{t}(e)$, i.e. we have $\mathrm{t}(e) = 1$ for a 1-edge e and $\mathrm{t}(e) = 0$ for a 0-edge e.

Lemma 1 Let v be a non-terminal BDD node v, the ε-value of which has lately been changed by a value $\Delta\varepsilon_v$. Further, assume the structure of the BDD has not changed above this node. Then the ε-value of an output node v' changes for every path $p = (v', e_{v'}, \ldots, u, e_u, v)$ by $\mathrm{pr}(p)\cdot\Delta\varepsilon_v$.

Proof. We have

$$\mathrm{pr}(p) = \mathrm{pr}\big(\mathrm{var}(v') = \mathrm{t}(e_{v'})\big)\cdot\ldots\cdot\mathrm{pr}\big(\mathrm{var}(u) = \mathrm{t}(e_u)\big) \tag{4}$$

for the path probability $\mathrm{pr}(p)$, since, for p to be chosen in an evaluation, all variables tested along p must be assigned the according Boolean values. Now the result can be seen straightforwardly by Eq. (2): the product of assignment probabilities on the right side of Eq. (4) is exactly the factor occurring before the term $\varepsilon(v)$ in an expression for $\varepsilon(v')$, derived with the developed, non-recurrent form of Eq. (2). □

We will call $\mathrm{pr}(p)$ the *weight* of v in v' along the considered path p from v' to v.

Of course we have to consider *all* paths from v' to v. To compute the total change $\Delta\varepsilon_{v'}$, we have to sum up the weights along all paths from v' to v. Let

$$\omega(v, v') := \sum_{\substack{p \text{ is path} \\ \text{from } v' \text{ to } v}} \mathrm{pr}(p) \tag{5}$$

[2] To keep the presentation simple, code handling the boundary case ($v = \mathbf{1}$) is omitted and equal probability of one and zero assignments for every variable is used for the algorithm in Fig. 2, i.e. $\mathrm{pr}(x = 0) = \mathrm{pr}(x = 1) = \frac{1}{2}$ for every variable x.

denote this total weight of v in v'. We have $\Delta\varepsilon_{v'} = \Delta\varepsilon_v \cdot \omega(v,v')$. By Eq. (5), $\omega(v,v')$ can also be interpreted as the overall probability of an evaluation reaching v from v'.

Next we want to express the weight of a non-terminal node v in the change of the global EPL-value for the considered BDD. This change is denoted $\Delta\varepsilon$ and the weight is denoted $\omega(v)$. In analogy to $\omega(v,v')$, the weight of v in another node v', $\omega(v)$ expresses the overall probability of an evaluation reaching v from an output node, i.e. $\omega(v) := \frac{1}{m}\sum_{i=1}^{m}\omega(o_i,v)$, where o_i is the output node representing f_i.

Note that by Eq. (5) the terms $\omega(o_i,v)$ still depend on an enumeration of all paths from v' to v.

However, it is possible to express $\omega(v)$ independent of path enumerations.

Lemma 2 Let v be a non-terminal node in a BDD F. Then the weight of v in the change of $\varepsilon(F)$ can be expressed as

$$\omega(v) = \frac{k}{m} + \sum_{\substack{v \text{ is } b\text{-son} \\ \text{of } v',\, b \in \mathrm{B}}} \mathrm{pr}\left(\mathrm{var}(v') = b\right) \cdot \omega(v'), \qquad (6)$$

where k is the number of single-output functions f_i which are represented by v (if any). **Proof.** We sketch the outline of an inductive proof with a case analysis. First, let v be a root node, i.e. v has no parent nodes. Then v must be an output node. By Eq. (3) can easily be seen that the contribution of an output node to the global change $\Delta\varepsilon$ must be $\frac{k}{m}$ (this also expresses the probability of an evaluation starting at output node v). This is correctly expressed by Eq. (6): the sum term vanishes, since v has no parent nodes.

Secondly, let v be an inner, non-output node. To reach v from any output node, first some parent node v' must be reached. Moreover, if v is a b-son of v', $\mathrm{var}(v')$ must be assigned b. Hence, the product of the according probabilities expresses the probability of an evaluation reaching v via v'. Note that $\omega(v')$ is the required weight resp. probability by induction hypothesis. Since we sum up these probabilities over all parent nodes, we obtain the total probability of an evaluation reaching v from an output node. From the previous, and since in Eq. (6) $\frac{k}{m}$ is added to this sum, the case of v being an inner output node is also handled correctly. □

Now we have the desired property $\Delta\varepsilon = \Delta\varepsilon_v \cdot \omega(v)$, i.e. a local change of ε_v is directly transferred to a change of the global ε-value. Next, we give an algorithm to compute the ω-values. Starting from the uppermost level, we can propagate the ω-values down from parent nodes to its child nodes which are residing on lower levels (see Fig. 3)[3]. In the next sections we will see that we do *not* have to use this algorithm after each variable swap. Hence, the locality of our approach is preserved.

C. Invariance of the Weights

The next results are crucial for the desired *locality* of our approach.

Lemma 3 Let F be a BDD representing a Boolean multi-output function f. Let v be a node on level k in F representing the Boolean function g (a cofactor of f with respect to the first $k-1$ variables in the ordering). Then the weight $\omega(v)$ can be expressed as follows:

$$\omega(v) = \sum_{\substack{b \in \mathbf{B}^{k-1} \\ \text{with } f_b = g}} \mathrm{pr}(x_1 = b_1) \cdot \ldots \cdot \mathrm{pr}(x_{k-1} = b_{k-1}). \qquad (7)$$

[3]Again, code handling the boundary case is omitted and we assume equal probability of one and zero assignments for every variable to keep the presentation simple.

```
/* assumes all values ω(v) are initially zero */
(1)  calc_omega(BDD F) {
(2)      for all nodes v representing output
         functions do {
(3)          ω(v) := ω(v) + 1/m;
(4)      }
(5)      for i := 1 to n do {
(6)          for all nodes v at level i do {
(7)              increase ω(then(v)) and
                 ω(else(v)) by 1/2 ω(v);
(8)          }
(9)      }
(10) }
```

Fig. 3. Iteratively computing the ω-values.

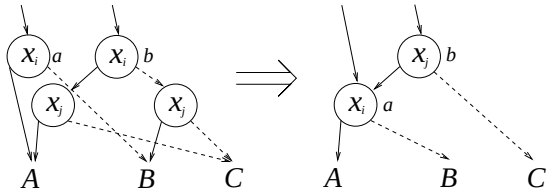

Fig. 4. A swap of two BDD levels i and j.

Proof. From the previous section we know that $\omega(v)$ expresses the probability of an evaluation reaching v from an output node. Eq. (7) straightforwardly expresses the same probability using the fact that every evaluation reaching a node defines a cofactor, which is functionally equivalent to the function represented by the node. □

Corollary 4 Let F be a BDD representing a Boolean function f and let v be a node in F. We assume fixed probabilities for the variable assignments to values in **B**. Then, if a) the function represented by v and b) the level of v are preserved with respect to a change in the variable ordering (and thus, in the graph structure) of the BDD, the value $\omega(v)$ also does not change, i.e. $\omega(v)$ is *invariant* with respect to this change.

Proof. In contrast to Eq. (6), Eq. (7) in Lemma 3 expresses $\omega(v)$ as a *function* of f, g and the level of v only. Since we consider a BDD representing a fixed f, the result follows. □

D. An Efficient Approach to Expected Path Length Minimization

In the previous section, we introduced an efficient method to directly transfer a local change in the ε-value of a node v to a change of the global ε-value of the BDD. This is possible with operations working only locally on the uppermost of two levels affected by a variable swap. In this section, we explain how to incorporate this technique into the final sifting modification.

Each swap of two adjacent levels i and j changes the local graph structure of the BDD in these two levels, leaving all other levels unchanged, e.g. see Fig. 4. For simplicity, assume equal probability of one and zero assignments for every variable. Before the swap, x_i resides in level i, x_j in level j. The local structural changes cause changes in the ε- and ω-values of nodes in the affected levels. E.g. see node a in Fig. 4: after the swap of levels i and j, node a has become a 1-son of node b. Thus we add $\frac{1}{2}\cdot\omega(b)$ to the old ω-value of a, updating

$\omega(a)$ following Eq. (6). Node b gets new child nodes after the swap. Hence $\varepsilon(b)$ is recalculated using the ε-values of the new child nodes (node a and the root node of subgraph C) following Eq. (2).

Also note that b still represents the same function and still is on the same level after the swap. Consequently, $\omega(b)$ remains unchanged by Corollary 4. The same invariance holds for all nodes outside the swapped levels. The change in the ε-value of the nodes v in level i is the value $\Delta\varepsilon_v$ described in Section B. The changes of ε for the child nodes of these nodes do not need to be transferred, since these changes already have been transferred to the parent nodes in level i.

Of course we here give only some of the cases to consider during a swap.

After having tried all positions by moving a variable x_i up and down, x_i is moved back to the best position seen in the previous movements. Let this position be on level k. To reestablish proper ε-values for subsequent movements of the other variables, all ε-values of nodes above level k are recalculated with a call to **eps_above_level**$(F, k-1)$ (see Fig. 2). This is an operation touching many levels, i.e. a large part of the whole BDD. But this only has to be done *once* after all swaps to determine the best position for a variable have been performed. This is possible since we can compute the global ε-value by transfer of the local changes while moving a variable up and down. Hence this still does not lead to a higher asymptotic time complexity than the complexity of the original sifting algorithm.

To summarize: a modification of the original sifting algorithm has been found which targets the expected path length in BDDs. This algorithm has the same asymptotic time complexity as the original sifting algorithm, as a sophisticated schema ensures that no traversal on the whole graph of the BDD is necessary when performing a variable swap. This schema consists of keeping track of local changes in the levels affected by the swap and then directly transferring these changes to a change in the global EPL value.

VI. Experimental Results

In this section we present our results applying the new approach, EPL sifting and maximal path length (MPL) sifting, i.e. a sifting modification targeting the classical criterion for circuit speed, the maximal delay on a critical path, as well as the standard sifting algorithm targeting BDD size to circuits of the LGSynth93 [13] benchmark set. A weaker form of MPL sifting with a simplified cost function has been used in [20]. Since we are interested in a validation of the quality of results yielded by EPL sifting, we implemented a sifting modification targeting the classical MPL criterion. As no approach with an only local behavior is known for MPL sifting, this algorithm has high run times. All algorithms have been integrated into the CUDD package [22] and were tested in the same system environment. We used a system with an Athlon CPU running at 1.4 GigaHz with a main memory of 1.5 GigaByte for our experiments. All methods use BDD size as second criterion in case of ties of the first criterion. In our tests, we used equal probabilities of one and zero assignments for every variable.

In a first series of experiments we compared MPL and EPL sifting. The results are given in Table II. In the first column the name of the function is given. *in* denotes the number of inputs of a function. Column *size* shows the initial size (given as number of BDD nodes) of the BDD representing the function. In columns *MPL* and *EPL* the maximal path length and the expected path length of the initial BDD for a function are given. The next three columns *size*, *MPL* and *EPL* give the size, the maximal path length and the expected path length for the BDD after applying the MPL sifting approach respectively. The next column *time* shows the runtime of the minimization by MPL sifting in CPU seconds. In the next four columns the same quantities size, MPL, EPL and runtime are shown for EPL sifting respectively.

TABLE I
Size, MPL, EPL and runtime sums for different criteria

criterion	∑ size	∑ MPL	∑ EPL	total time
initial	391155	821	189.04	–
EPL	181842	753	133.58	275.30s
MPL	169408	753	156.31	2272.37s
size	168914	777	162.85	47.30s

In a second series of experiments we also applied "classical" BDD size-driven sifting to our set of test cases. The accumulated results and total run times of the two series of experiments are given in Table I. The first column *criterion* gives the optimization criterion targeted by the used method (here, the entry "initial" means the BDDs before any method has been applied). Columns two to four each state the sum of a BDD characteristic over the BDDs for all test cases respectively: in column two, this characteristic is BDD size, in column three it is MPL and in column four it is EPL. The last column states the total runtime for the whole test suite (the "-" for row *initial* means no optimization has been carried out).

As the results show, EPL sifting achieves almost the same reduction in MPL as MPL sifting: both EPL and MPL sifting improve the initial MPL by 8.3% in average and the improvement can be up to 48.9% (e.g., see *dalu*). EPL sifting is also the fastest delay-driven minimization approach preserving high quality of results: the total time (average) speed up factor of EPL sifting compared to MPL sifting is 8.25 and can be up to two orders of magnitude (e.g., see *i7*).

Moreover, EPL sifting also yields the best starting point for critical path analysis, as the resulting EPL values (corresponding to the average gate delay) are much better than for the other minimization methods. EPL sifting reduces the initial EPL by 29.3% on average. The improvement can be up to 63.4%, especially for larger instances (e.g., see *dalu*). EPL sifting achieves an EPL which is better by 14.5% on average compared to the EPL yielded by MPL sifting. The improvement over MPL sifting can be even higher, up to 40.1% (e.g., see *x4*).

All methods also improve BDD size significantly. In this, EPL sifting yields only 7.3% more size than MPL sifting (and "classical" size sifting) on average. This allows higher circuit speed at only low area overhead.

VII. Conclusions and Future Work

We presented an efficient technique to optimize BDDs with respect to the expected path length. This criterion measures the evaluation time for functional simulation. It also models average gate delay assuming a unit delay model.

The method is based on sifting and uses a new sophisticated approach to keep track of local changes and their impact on the global change during a variable swap. Thus it achieves small run times, staying within the time complexity of the original sifting algorithm. In previous approaches a comparable speed can only be achieved by simplification of the target function, possibly leading to poorer results.

Experimental results are reported demonstrating that the proposed technique also results in BDDs highly optimized for the synthesis of high speed PTL circuits. It achieves almost the same quality of results as a sifting modification exactly targeting the classical criterion of maximal path length. Reductions in the length of critical paths of up to 48.9% have been observed. Moreover, our approach yields BDDs which are a

TABLE II
COMPARISON OF MPL AND EPL SIFTING

name	in	size	MPL	EPL	MPL sifting				EPL sifting			
					size	MPL	EPL	time	size	MPL	EPL	time
apex6	135	2759	21	3.56	639	20	2.37	6.59s	662	20	2.33	0.17s
apex7	49	1659	24	5.03	310	19	2.59	0.54s	274	19	2.25	0.05s
b9	41	177	13	3.17	107	13	3.04	0.18s	125	13	2.65	0.02s
c1355	41	43869	41	30.76	30326	41	26.70	126.11s	39201	41	20.82	64.97s
c3540	50	223227	30	10.94	73319	30	10.66	1171.64s	46228	30	9.81	131.01s
c499	41	39377	41	29.65	30459	41	27.41	369.72s	38465	41	20.34	54.71s
c5315	178	5247	47	4.00	2611	47	3.89	127.28s	3137	47	4.07	1.44s
c880	60	15544	42	5.65	4865	41	5.27	25.09s	9883	41	4.82	3.49s
cht	47	149	5	2.71	89	4	2.63	0.17s	124	4	2.06	0.02s
dalu	75	12946	47	16.05	1216	24	5.88	24.73s	1006	24	4.94	3.13s
example2	85	468	16	2.54	298	14	2.71	1.28s	393	14	2.18	0.06s
frg2	143	2230	22	3.00	1434	20	2.42	15.98s	1648	20	2.32	0.37s
i3	132	132	32	4.46	132	32	4.46	4.57s	132	32	4.46	0.08s
i4	192	420	47	6.56	300	47	4.78	15.52s	300	47	4.38	0.21s
i5	133	311	19	2.50	133	19	1.98	5.32s	133	19	1.98	0.08s
i6	138	412	4	3.10	208	4	3.24	4.69s	274	4	3.05	0.07s
i7	199	504	4	3.25	367	4	3.35	15.10s	434	4	3.18	0.13s
i8	133	3980	16	5.42	1678	13	3.49	25.91s	2495	13	3.40	0.81s
i9	88	2270	12	5.48	1659	12	5.00	5.48s	1821	10	4.96	0.21s
k2	45	2012	24	5.03	1355	24	4.03	1.23s	1438	24	4.01	0.14s
pair	173	13845	49	6.08	5857	49	4.13	197.76s	8775	49	3.76	4.40s
rot	107	8322	57	4.27	6374	57	4.05	48.82s	15062	59	3.08	7.69s
s5378	199	5218	41	3.70	2489	33	3.43	65.50s	5975	34	3.11	1.43s
s641	54	1351	27	3.69	628	25	3.14	0.91s	724	26	2.65	0.11s
s713	54	1351	27	3.69	628	25	3.14	0.92s	724	26	2.65	0.10s
s838.1	66	244	55	4.08	298	38	3.37	0.98s	625	35	2.92	0.10s
x1	51	1296	23	3.88	487	22	2.80	0.75s	603	22	2.67	0.07s
x3	135	945	20	3.00	612	20	2.36	7.14s	669	20	2.34	0.14s
x4	94	890	15	3.79	530	15	3.99	2.46s	512	15	2.39	0.07s

better starting point for an additional critical path analysis (by up to 40.1%), and is faster than maximum path length sifting by up to two orders of magnitude.

It is focus of current work to transfer the techniques presented here to more complex delay models and to further investigate the impact of the distribution of assignment probabilities on the quality of results.

ACKNOWLEDGEMENTS

The authors would like to thank Stefan Höhne who contributed to this work during an undergraduate course at the University of Kaiserslautern.

REFERENCES

[1] P. Ashar and S. Malik, "Fast functional simulation using branching programs," in *Int'l Conf. on CAD*, pp. 408–412, 1995.

[2] H. Bakoglu, *Circuits, Interconnections and Packaging for VLSI*, Addison-Wesley, 1990.

[3] L. Benini, E. Macii, and M. Poncino, "Telescopic units: increasing the average throughput of pipelined designs by adaptive latency control," in *Design Automation Conference*, pp. 22–27, 1997.

[4] K. Brace, R. Rudell, and R.E. Bryant, "Efficient Implementation of a BDD Package", in *Design Automation Conf.*, pp. 40-45, 1990.

[5] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," in *IEEE Trans. on Comp.*, Vol. 8(35), pp. 677–691, 1986.

[6] R. Drechsler, N. Drechsler, and W. Günther, "Fast exact minimization of BDDs," in *IEEE Trans. on CAD*, Vol. 3(19), pp. 384–389, 2000.

[7] R. Drechsler and W. Günther, *Towards One-Pass Synthesis*, Kluwer Academic Publishers, 2002.

[8] R. Ebendt, W. Günther, and R. Drechsler, "Combination of lower bounds in exact BDD minimization," in *Design, Automation and Test in Europe*, pp. 758–763, 2003.

[9] G. Fey and R. Drechsler, "Minimizing the number of paths in BDDs," in *15th Symp. on Integrated Circuits and System Design*, pp. 359–364, 2002.

[10] Y. Iguchi and T. Sasao and M. Matsuura, "Evaluation of multiple-output logic functions using decision diagrams," in *Asian South Pacific Design Automation Conference*, pp. 312-315, 2003.

[11] N. Ishiura, H. Sawada, and S. Yajima, "Minimization of binary decision diagrams based on exchange of variables," in *Int'l Conf. on CAD*, pp. 472–475, 1991.

[12] Y.-M. Jiang, A. Krstic, K.-T. Cheng, and M. Marek-Sadowska, "Post-layout logic restructuring for performance optimization," in *Design Automation Conference*, pp. 662–665, 1997.

[13] Collaborative Benchmark Laboratory, *1993 LGSynth Benchmarks*, North Carolina State University, Department of Computer Science, 1993.

[14] T. H. Liu, A. Aziz, and J. Burns, "Performance driven synthesis for pass-transistor logic," in *Int'l Workshop on Logic Synth.*, pp. 255–259, 1998.

[15] Y. Y. Liu, K. H. Wang, T. T. Hwang, and C. L. Liu, "Binary decision diagram with minimum expected path length," in *Design, Automation and Test in Europe*, pp. 708–712, 2001.

[16] L. Macchiarulo, L. Benini, and E. Macii, "On-the-fly layout generation for PTL macrocells," in *Design, Automation and Test in Europe*, pp. 546–551, 2001.

[17] A. Mukherjee, R. Sudhakar, M. Marek-Sadowska, and S. Long, "Wave steering in YADDs: a novel non-iterative synthesis and layout technique," in *Design Automation Conf.*, pp. 466–471, 1999.

[18] S. Nagayama, A. Mishchenko, T. Sasao, and J. Butler, "Minimization of Average Path Length in BDDs by Variable Reordering," in *Proc. of International Workshop on Logic and Synthesis 2003*, Los Angeles, 2003.

[19] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," in *Int'l Conf. on CAD*, pp. 42–47, 1993.

[20] C. Scholl and B. Becker, "On the generation of multiplexer circuits for pass transistor logic," in *Design, Automation and Test in Europe*, pp. 372–378, 2000.

[21] C. Scholl, R. Drechsler, and B. Becker, "Functional simulation using binary decision diagrams," in *Int'l Conf. on CAD*, pp. 8–12, 1997.

[22] F. Somenzi, *CU Decision Diagram Package Release 2.3.1*, University of Colorado at Boulder, 2002.