to be published in
Proc. of 5th Int. Workshop on Applications of the Reed-Muller Expansion in Circuit Design (Reed-Muller 2001),
Starkville, Mississippi, USA, 10-11 August 2001, pp. 51-56.

# Improving XOR-Node Placement for ⊕-OBDDs

Christoph Meinel, Harald Sack
*FB IV - Informatik, Universität Trier*
*D-54286 Trier, Germany*
*{meinel,sack}@uni-trier.de*

## Abstract

*Ordered Binary Decision Diagrams (OBDDs) have already proved useful in the process of electronic design automation. Due to limitations of the descriptive power of OBDDs more general models of Binary Decision Diagrams have been studied. In this paper, ⊕-OBDDs as a true extension of the OBDD data structure are addressed. One important factor for the representation size of ⊕-OBDDs is determined by the number and the position of the introduced ⊕-nodes. Based on a simple greedy strategy that switches between different function decompositions, it is shown how to introduce ⊕-nodes during ⊕-OBDD synthesis just at the right place. The efficiency of the approach is proven by symbolic simulation of standard benchmarks.*

## 1. Introduction

A major problem in the computer aided design of digital circuits is the choice of a suitable representation of the circuit functionality for the computer's internal use. A concise representation, which simultaneously provides the possibility of fast manipulation is very important for all problems given in terms of switching functions. During the last decade, Ordered Binary Decision Diagrams (OBDDs) have proved to be well qualified for this purpose (for an overview see [13]).

But, the descriptive power of OBDDs is limited, due to their property of being a canonical representation for Boolean functions. On the one hand, this important quality is responsible for the nice algorithmic properties of OBDDs. But, on the other hand, the OBDD representation for most Boolean functions must be of exponential size w.r.t. the number of input variables, and not every Boolean function of practical importance can be represented efficiently. E.g., the OBDD-representations of the *multiplication* or the *hidden weighted bit function* are always of exponential OBDD-size [4] independent of the chosen order of the input variables. For this reason, generalizations of the OBDD data structure have been studied.

In this paper we address ⊕-OBDDs (also known as Mod2-OBDDs), a true extension of OBDDs [6]. ⊕-OBDDs are more, sometimes even exponentially more, space-efficient than OBDDs are. They preserve the algorithmic properties of OBDDs: important operations as *apply, quantification*, and *composition* have the same complexity as in the case of OBDDs. Even better, the Boolean functions *exclusive or* (XOR) and *logical equivalence* (EQU) can be performed in constant time.

However, ⊕-OBDDs do not provide a canonical representation of Boolean functions and therefore, equivalence can only be tested fast, if probabilistic techniques are applied [6]. A deterministic equivalence test requires time $O(|P|^3)$ [16], with $|P|$ denoting the number of nodes of the ⊕-OBDD $P$, and thus, it is way too slow for any application in practice.

The representation size of ⊕-OBDDs does not only depend on the chosen variable order as in the case of OBDDs, but it also depends on number and position of ⊕-nodes. To make use of the full potential of ⊕-OBDDs, also the number and the positions of the ⊕-nodes is rather important [12].

In this paper we introduce a heuristic for deciding, where ⊕-nodes should be positioned during ⊕-OBDD synthesis. The introduction of new ⊕-nodes can be achieved by employing alternative function decompositions that are depending on the XOR-operation, as, e.g. the positive (negative) Davio expansion [14, 15]. For deciding, when to apply the alternative decomposition, a threshold value is fixed before each synthesis step. Only, if the result of the regular synthesis algorithm exceeds the given threshold value in size, the alternative decomposition is applied and new ⊕-nodes can be introduced.

For giving proof about the efficiency of this simple heuristic, it is applied to symbolic simulation of a set of standard benchmarks [9].

The paper is structured as follows: In Section 2, we recall basic definitions concerning ⊕-OBDDs. Section 3 covers the synthesis algorithm for ⊕-OBDDs and recalls the alternative function decompositions. Section 4 introduces our heuristic for smart ⊕-node placement. Section 5 concludes with a discussion of achieved experimental results.

## 2. $\oplus$-OBDDs - an Overview

### Definition of the Data Structure

A $\oplus$-OBDD $P$ over a set $X_n = \{x_1, \ldots, x_n\}$ of Boolean variables is a directed acyclic connected graph $P = (V, E)$. $V$ is the set of nodes, consisting of non-terminal nodes with out-degree 2, and of terminal nodes with out-degree 0. There is a distinguished non-terminal node, the *root*, which, as only node, has the in-degree 0. The two terminal nodes with no outgoing arcs are labeled with the Boolean constants 0 and 1. The remaining nodes are either labeled with Boolean variables $x_i \in X_n$ (*branching nodes*), or with the binary Boolean operator XOR ($\oplus$*-nodes*). On each path, every variable must occur at most once. In the following, let $l(v)$ denote the label of the node $v \in V$ and $|P|$ the number of non terminal nodes of $P$.

$E \subseteq V \times V$ denotes the set of edges. The two edges starting in a branching node $v$ are labeled with 0 and 1. The *0(1)-successor* of node $v$ is denoted by $v_0(v_1)$. There is a permutation $\pi$, which defines an order $x_{\pi(1)} < x_{\pi(2)} < \ldots < x_{\pi(n)}$ on the set of input variables. If $w$ is a successor of $v$ in $P$ with $l(v), l(w) \in X_n$, then $l(v) < l(w)$ according to $\pi$ must hold.

Note that since the $\oplus$-operation is symmetric, the outgoing edges of $\oplus$-nodes do not have to be labeled separately. The function $f_P$ associated with the $\oplus$-OBDD $P$ is determined in the following way: For a given input assignment $a = (a_1, \ldots, a_n) \in \{0,1\}^n$, the Boolean values assigned to the leaf nodes are extended to all other nodes of $P$ as follows:

- Let $v_0$ and $v_1$ be the successors of $v$, carrying the Boolean values $\delta_0, \delta_1 \in \{0,1\}$.
- If $v$ is a branching node, $l(v) = x_i \in X_n$, then $v$ is associated with $\delta_{a_i}$.
- If $v$ is a $\oplus$-node, then $v$ is associated with $\oplus(\delta_0, \delta_1) = (\delta_0 + \delta_1) \bmod 2$.

The function $f_P(a)$ computes to the value associated with the source of $P$.

To achieve a more compact representation, we may furthermore consider the use of complemented edges [1, 10]. $\oplus$-OBDDs are also a generalization of Kronecker Functional Decision Diagrams (KFDDs) or pseudo Kronecker Functional Decision Diagrams (pKFDDs), but, they provide a more compact representation than KFDDs or pKFDDs do [6].

$\oplus$-OBDDs do not provide a canonical representation of Boolean functions, i.e. there might be several different representations $P_f^1, \ldots P_f^k$, $k \in \mathbb{N}$ for the same Boolean function $f$. Thus, testing the equivalence of two $\oplus$-OBDDs becomes a rather difficult and important task.

### Probabilistic Equivalence Test

Since a deterministic equivalence test for $\oplus$-OBDDs requires runtime $O(|P|^3)$ [16], for practical applications we have to choose a faster method. A probabilistic equivalence test for $\oplus$-OBDDs as proposed in [5] requires only linearly many arithmetic operations in the number of variables. It is based on a probabilistic equivalence test for *read-once branching programs* (BP1), originally introduced in [2] and further refined in [8]. Equivalence of two $\oplus$-OBDDs is determined probabilistically, after an algebraic transformation of the $\oplus$-OBDDs in terms of polynomials over a finite field. For a more detailed description of this algorithm see [12].

### Reduction Rules for $\oplus$-OBDDs

The reduction rules that are already known for OBDDs and, if exhaustively applied, guarantee a canonical representation for OBDDs, have to be extended for $\oplus$-OBDDs. Here, these reduction rules serve only for a reduction in size, but they are not able to provide canonicity for $\oplus$-OBDDs. In addition to the regular reduction rules for OBDDs that can be applied to the branching nodes of a $\oplus$-OBDD, reductions for $\oplus$-nodes have to be considered according to the node's functionality[12].

## 3. Synthesis of $\oplus$-OBDDs

For the synthesis of $\oplus$-OBDDs, the already known *ITE*-algorithm [3] that is applied for OBDDs can easily be extended.

To connect two Boolean functions $f$ and $g$ given in terms of OBDDs with an arbitrary Boolean operation $\otimes$ the *Boole-/Shannon decomposition (BS)* w.r.t. variable $x_i$ is applied:

$$f \otimes g = x_i(f|_{x_i} \otimes g|_{x_i}) + \overline{x_i}(f|_{\overline{x_i}} \otimes g|_{\overline{x_i}}).$$

$f_{x_i}$ denotes the positive cofactor of the Boolean function $f$, where the input variable $x_i$ is substituted with the constant $x_i = 1$. $f_{\overline{x_i}}$ denotes the negative cofactor of $f$, respectively. The composition of the cofactors can be computed recursively. For efficiency reasons all Boolean operations are mapped to a single general operation, which is able to express all Boolean operations, the so called *If-Then-Else operator (ITE)* [3]. $ITE(x, y, z)$ is a three parameter function computing *if x, then y, else z*:

$$ITE(x, y, z) = x \cdot y + \overline{x} \cdot z$$

For computing the synthesis of functions $f, g, h$ represented as OBDDs, ITE is called recursively w.r.t. the top variable $x_i$ of the involved OBDDs.

$$\begin{aligned} ITE(f,g,h) &= (x_i, \quad ITE(f|_{x_i}, g|_{x_i}, h|_{x_i}), \\ &\qquad\quad ITE(f|_{\overline{x_i}}, g|_{\overline{x_i}}, h|_{\overline{x_i}})) \end{aligned}$$

The recursion stops, if the first argument is constant, if the second and the third arguments are constant, or if the second and the third arguments are equal.

For $\oplus$-OBDDs, as an extension for computing $f \oplus g$ or $f \equiv g$, a new $\oplus$-node will be created and connected to $f$ and $g$, where $f \equiv g = \overline{f \oplus g}$. In all other cases, the regular *ITE*-algorithm is applied with an adapted cofactor creation algorithm for $\oplus$-OBDDs, where, for the computation of the cofactor $f_{x_i}$ of a function $f$ associated with a $\oplus$-node $v_f$ according to a variable $x_i$, in some cases, the allocation of a new $\oplus$-node $v_{f_{x_i}}$ is required that is connected to the cofactors of the left and right successor of $v_f$ [12]. The extended ITE-algorithm for $\oplus$-OBDDs is denoted as ITE-$\oplus$-algorithm.

But, for symbolic simulation, if the circuit under consideration does not contain any XOR(EQU) gate, the ITE-$\oplus$-algorithm would just create only an OBDD instead on a $\oplus$-OBDD. To benefit from the potential of $\oplus$-OBDDs, somehow, $\oplus$-nodes have to be introduced into the data structure. This can be achieved by employing alternative function decompositions based on the application of XOR, as e.g., the positive or negative Davio expansion (pDE/nDE), also referred to as Reed-Muller expansion [14, 15]. There, the XOR operator can directly be mapped to a $\oplus$-node.

$$\begin{aligned} \text{pDE: } f &= f|_{\overline{x_i}} \oplus x_i(f|_{x_i} \oplus f|_{\overline{x_i}}) \\ \text{nDE: } f &= f|_{x_i} \oplus \overline{x_i}(f|_{x_i} \oplus f|_{\overline{x_i}}). \end{aligned}$$

The synthesis algorithm for $\oplus$-OBDDs based on pDE-/nDE-decomposition is denoted as APPLY-$\oplus$-algorithm.

## 4. A Heuristic for $\oplus$-Node Placement

As shown in [12], not only the chosen variable order, but also the number and the position of the introduced $\oplus$-nodes determines heavily the size of $\oplus$-OBDDs.

As an indicator, whether the introduction of a $\oplus$-node at a specific place in a $\oplus$-OBDD might be useful or not, we consider the satisfaction of the following assumption: The introduction of a $\oplus$-node is useful, if it results in a $\oplus$-OBDD of smaller size. Now, $\oplus$-nodes can be positioned randomly into the already constructed $\oplus$-OBDD and we decide, whether to keep them or not according to their effect on the $\oplus$-OBDD size. But, this approach requires the construction of the complete $\oplus$-OBDD first, before we have the possibility to improve its size. In consequence, we might construct only a part of the $\oplus$-OBDD, introduce a satisfactory number of $\oplus$-nodes, and afterwards, continue with its construction.

```
Input: ⊕-OBDD $P_f$, $P_g$, and operator ⊗
Output: ⊕-OBDD $P_{res}$, representing $res = f \otimes g$

local_greedy_synthesis($P_f$, $P_g$, ⊗) {
   res-ite = ITE-⊕($P_f$, $P_g$, ⊗);
   if ( size(res-ite) > threshold ) {
     res-alt = APPLY-⊕($P_f$, $P_g$, ⊗);
     if ( res-alt < res-ite ) {
       res = res-alt;
       delete res-ite;
     } else {
       res = res-ite;
       delete res-alt;
     }
   }
   return($res$);
}
```

**Figure 1**: Heuristic for $\oplus$-OBDD Node Placement.

By following this concept, we end up in a dynamic approach, which in each construction step of the $\oplus$-OBDD compares its size with and without introduced $\oplus$-node. In symbolic simulation of a combinatorial design this means that for each single gate $G$, we construct the $\oplus$-OBDD $P_G$ representing the function $f_G$ of $G$. First, we are using the ITE-algorithm and construct $P_{f-ite}$, and additionally employ either pDE(nDE)-APPLY-$\oplus$-algorithm, resulting in $P_{f-nDE}$ ($P_{f-pDE}$). Next, we compare the two $\oplus$-OBDD sizes and decide, which $\oplus$-OBDD to keep. If $|P_{f-ite}| > |P_{f-nDE}|$ ($|P_{f-pDE}|$), then we keep $P_{f-nDE}$ ($P_{f-pDE}$) and vice versa.

Thus, locally we always try to make use of the smallest possible $\oplus$-OBDD. But, of course this is only a local minimum. Another disadvantage is that for each synthesis step, we have always to construct both versions of the $\oplus$-OBDD with the two synthesis procedures resulting in a significant runtime overhead. To increase the efficiency of the approach, we limit the construction of the alternative $\oplus$-OBDDs to the case, only when the regular ITE-algorithm computes a $\oplus$-OBDD of a size that is passing a certain fixed threshold. Thus, the additional construction of $\oplus$-OBDDs is limited to the cases, when an alternative representation can be of a major advantage. For an outline of this locally greedy algorithm see Fig. 1.

An important factor for this heuristic is of course the proper choice of the threshold value. For our experiments, we have chosen from the following possibilities:

- Set the threshold value to the maximum size of the two operands multiplied by a constant $c$, thus $t = c \cdot \max(|P_f|, |P_g|)$ (MAX).
- Set the threshold value to the sum of the sizes of the two operands multiplied by a constant $c$, thus $t = c \cdot (|P_f| + |P_g|)$ (ADD).

| OBDD-size | | $\oplus$-OBDD size | | | |
|---|---|---|---|---|---|
| OBDD | % | pDE | % | nDE | % |
| 4.468.873 | 100 | 3.261.714 | 73 | 4.4687.023 | 104.9 |

**Table 1:** Reference Table for OBDDs and $\oplus$-OBDDs with pDE/nDE.

## 5. Experimental Results and Conclusion

For showing the efficiency of the heuristic, we have chosen the symbolic simulation of a subset of the LGSynth'93 [9] benchmarks. All experiments are computed on an Intel Pentium III 500 MHz based Linux system. Memory size is limited to 200 MB and computation time to 2 CPU hours. Circuits that are resulting in OBDDs with less than 100 nodes or that are exceeding the given resource limitations are excluded. The variable order for all circuits was kept fixed for showing the effect of dynamic $\oplus$-node placement and reflects the order of the inputs given with the circuit description. For the probabilistic equivalence test it would have been sufficient to limit the number of signatures, i.e. the number of independent probabilistic equivalence tests, used for identifying $\oplus$-OBDDs to $n = 2$, but for reasons of security $n = 3$ was chosen.

In Table 2 the results of these experiments are put together. For different constant factors $0.6 \leq c \leq 2.0$ we have listed the overall size achieved for all benchmarks for the methods denoted as MAX, with application of nDE/pDE, and as ADD, as referred in the list above. Additionally we have also tried to reverse the decision criteria for the heuristic, i.e. we use nDE/pDE as default function decomposition and only in the case, when the given threshold value is exceeded, we switch to the ITE-$\oplus$ algorithm with the regular Boole/Shannon-decomposition. For this strategy (further denoted as *pDE-first/nDE-first*), where much more $\oplus$-nodes are created, the achieved overall sizes are worse compared to the original approach and only the results for the best choice of the threshold parameter $c = 1.0$ is listed for that case. This fact confirms the results achieved in [12] that a small number of $\oplus$-nodes placed at well chosen positions inside the $\oplus$-OBDD provides the best overall effect in the average. For a reference in Table 1 the overall sizes for OBDDs and for exclusive application of pDE and nDE are also listed. The values given in percentages are always referring to the OBDD size, which is denoted as 100%. For ADD we have only listed the achieved sizes for nDE, because for pDE the results are only slightly different. For a complete overview of the achieved results for the single benchmark circuits and MAX-nDE see Table 3.

By comparing the overall achieved size, the first thing to state is that the exclusive application of pDE results in 27% gain in size, compared to a 5% loss for nDE, if related to the original OBDD size. By applying the locally greedy heuristic with different constant parameter $0.6 \leq c \leq 2.0$, we can see that the size is minimal for choosing the parameter $c \approx 1.0$. There, we are able to achieve an up to 33% win for the overall size, which is better compared to the exclusive application of nDE or pDE. In their general behavior the two approaches MAX and ADD produce only slight differences in size.

For circuits that benefit from the introduction of $\oplus$-nodes, the exclusive application of nDE/pDE is often better than the proposed heuristics. But, for circuits that don't benefit from the introduction of $\oplus$-nodes, the heuristic is often much better than the exclusive application of pDE/nDE. For the circuit $mult16a$, e.g. the heuristic is always producing a smaller result compared to the OBDD size or the exclusive application of nDE/pDE. But, on the other hand, for $cm150a$, the heuristic is not able to reproduce a result of similar quality as the exclusive application of pDE/nDE. For all methods and all parameters the achieved $\oplus$-OBDD size is approximately of the size of the OBDD or even better.

Thus, in general our heuristic is able to keep the benefits of both decompositions, BS and nDE/pDE. For the heuristic the runtime increases in the average of about 10%-15% in comparison to the average runtime for standard $\oplus$-OBDD synthesis. But, considering the general reduction in size, spending this additional amount of time is worth while.

The achieved results could be further improved by changing the positions of the already introduced $\oplus$-nodes. Exchanging a branching node with an adjacent $\oplus$-node effects the $\oplus$-OBDD only locally and thus, can be computed rather fast [7, 11]. Based on this technique, in combination with the proposed algorithm, new heuristics can be developed for further $\oplus$-OBDD

| | ⊕-OBDD Size | | | | | |
|---|---|---|---|---|---|---|
| $c$ | MAX (pDE) | % | MAX (nDE) | % | ADD | % |
| 0.6 | 3.634.456 | 81.3 | 3.216.892 | 72.0 | 3.214.424 | 71.9 |
| 0.7 | 3.633.394 | 81.3 | 3.215.694 | 72.0 | 3.212.698 | 71.9 |
| 0.8 | 3.469.411 | 77.6 | 3.212.683 | 71.9 | 3.218.253 | 72.0 |
| 1.0 | 3.213.905 | 71.9 | **2.997.931** | 67.1 | **3.008.490** | 67.3 |
| 1.2 | **3.000.038** | 67.1 | 2.999.501 | 67.2 | 3.009.105 | 67.3 |
| 1.5 | 3.000.179 | 67.1 | 3.009.641 | 67.3 | 3.011.906 | 67.4 |
| 2.0 | 3.013.619 | 67.4 | 3.015.341 | 67.5 | 3.016.222 | 67.5 |
| $c$ | MAX (pDE-first) | % | MAX (nDE-first) | % | ADD | % |
| 1.0 | 4.029.754 | 90.2 | 4.202.453 | 94.0 | 4.208.246 | 94.2 |

**Table 2:** Heuristic for ⊕-Node Placement - Overall Results.

minimization.

## References

[1] S. B. Akers, Binary Decision Diagrams, *in IEEE Trans. on Computers*, vol. **c-27**, no. 6, 1978, 509-516.

[2] M. Blum, A. K. Chandra, M N. Wegman, Equivalence of Free Boolean Graphs can be decided Probabilistically in Polynomial Time, *in Information Processing letters* **10**, No. 2, 1980, 80-82.

[3] K. S. Brace, R. L. Rudell, R. E. Bryant, Efficient Implementation of a BDD Package, *in Proc. of the 27th ACM/IEEE Design Automation Conf.*, 1990, 40-45.

[4] R. E. Bryant, On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Applications to Integer Multiplication, *in IEEE Trans. on Computers* **40** Vol. 2, 1991, 205-213.

[5] J. Gergov, Ch. Meinel, Frontiers of Feasible and Probabilistic Feasible Boolean Manipulation with Branching Programs, *in Proc. 10th Annual Symp. on Theoretical Aspects of Computer Science*, **665** of LNCS, Springer, 1993, 576-585.

[6] J. Gergov, C. Meinel, Mod2-OBDDs: A Data Structure that generalizes EXOR-sum-of-products and Ordered Binary Decision Diagrams, *in Formal Methods in System Design* **8**, Kluwer, 1996, 273-282.

[7] A. Hett, R. Drechsler, B. Becker, Reordering Based Synthesis, *in Proc. of the 3rd Int. Workshop on Applications of the Reed-Muller Expansion in Circuit Design (RM'97)*, Oxford, UK, 1997, 13-22.

[8] J. Jain, M. Abadir, J. Bitner, D. S. Fussell, J. A. Abraham, IBDDs: An Efficient Functional Representation for Digital Designs, *in Proc of the European Conference on Design Automation* (1992), 440-446.

[9] LGSynth93 Benchmarks: `http://www.cbl.ncsu.edu/CBL_Docs/lgs91.html`.

[10] J.-C. Madre, J.-P. Billon, Proving Circuit Correctness using Formal Comparison between Expected and Extracted Behaviour, *in Proc. 25th ACM/IEEE Design Automation Conference (Anaheim, CA)*, 1988, 205-210.

[11] C. Meinel, H. Sack, Algorithmic Considerations of ⊕-OBDD Reordering, *in Proc. of the 4th International Workshop on Applications of the Reed-Muller Expansion in Circuit Design (Victoria, BC, Canada)*, 1999, 197-184.

[12] C. Meinel, H. Sack, Mod2OBDDs - a BDD Structure for Probabilistic Verification, *in Electronic Notes in Theoretical Computer Science*, vol.**22**, 2000.

[13] Ch. Meinel, T. Theobald, Algorithms and Data Structures in VLSI Design: OBDD - Foundations and Applications, *Springer*, Heidelberg, 1998.

[14] D. E. Muller, Application of Boolean Algebra to Switching Circuit Design and Error Detection, *in IRE Trans. on Electronic Computing* **EC-3**, 1954, 6-12.

[15] L. S. Reed, A Class of Multiple Error-Correcting Codes and their Decoding Scheme, *in IRE Trans. on Information Theory* **4**, 1954, 38-42.

[16] S. Waack, On the Descriptive and Algorithmic Power of Parity Ordered Binary Decision Diagrams, *in Proc. 14th Symp. on Theoretical Aspects of Computer Science*, **1200** of LNCS, Springer, 1997.

|  |  | $\oplus$-OBDD Size |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|
|  |  |  | Threshold Factor $c$ |  |  |  |  |  |
| **Circuit** | **OBDD** | **nDE** | **0.6** | **0.8** | **1.0** | **1.2** | **1.5** | **2.0** |
| sbc | 3715 | 4598 | 3757 | 3755 | 3792 | 3785 | 3775 | 3767 |
| s967 | 1732 | 1073 | 1683 | 1683 | 1655 | 1642 | 1631 | 1640 |
| s820 | 2651 | 552 | 1642 | 1642 | 1638 | 1634 | 2117 | 2651 |
| s713 | 1352 | 3554 | 1433 | 1433 | 1408 | 1386 | 1366 | 1355 |
| s641 | 1352 | 3550 | 1423 | 1423 | 1398 | 1376 | 1360 | 1355 |
| s635 | 656 | 1746 | 798 | 793 | 660 | 660 | 659 | 655 |
| s526 | 232 | 371 | 269 | 267 | 259 | 246 | 229 | 231 |
| s510 | 19076 | 636 | 9738 | 9738 | 9742 | 9740 | 9713 | 9924 |
| s499 | 336 | 640 | 570 | 570 | 337 | 337 | 337 | 356 |
| s444 | 226 | 390 | 246 | 246 | 235 | 232 | 225 | 225 |
| s420 | 262227 | 732 | 262334 | 262334 | 262311 | 262245 | 262227 | 262227 |
| s386 | 281 | 295 | 273 | 255 | 262 | 260 | 271 | 280 |
| s3271 | 3365 | 6437 | 4175 | 3865 | 3541 | 3498 | 3403 | 3369 |
| s208 | 1033 | 186 | 1054 | 1054 | 1049 | 1049 | 1033 | 1033 |
| s1512 | 18896 | 10941 | 18763 | 18762 | 18746 | 18727 | 18690 | 18729 |
| s1494 | 1016 | 1378 | 1287 | 1287 | 1073 | 1078 | 1070 | 1015 |
| s1488 | 1016 | 1316 | 1238 | 1200 | 1030 | 1033 | 1023 | 1015 |
| s1423 | 98454 | 134008 | 99836 | 98531 | 98462 | 98460 | 98415 | 98519 |
| s1269 | 48176 | 39922 | 50769 | 49966 | 49072 | 49068 | 48221 | 48211 |
| s1196 | 2294 | 3844 | 2341 | 2341 | 2353 | 2341 | 2291 | 2263 |
| rot | 166674 | 266795 | 161506 | 163788 | 166825 | 166757 | 166705 | 166700 |
| mult16a | 360442 | 655125 | 163839 | 163839 | 163838 | 163838 | 163838 | 163838 |
| mm9b | 848081 | 658964 | 264856 | 264856 | 264856 | 264856 | 264838 | 264838 |
| mm9a | 735768 | 533707 | 220374 | 220374 | 220374 | 220374 | 220356 | 220356 |
| comp | 458698 | 859988 | 544543 | 544543 | 544541 | 544539 | 544549 | 544539 |
| mm4a | 675 | 1439 | 680 | 683 | 671 | 671 | 674 | 674 |
| dsip | 13921 | 9675 | 7722 | 13082 | 7717 | 7715 | 13923 | 13920 |
| x3 | 2760 | 2369 | 2625 | 2495 | 2429 | 2428 | 2762 | 2761 |
| x1 | 1297 | 2948 | 1358 | 1336 | 1331 | 1329 | 1307 | 1298 |
| vg2 | 1044 | 2071 | 1120 | 1107 | 1050 | 1045 | 1042 | 1042 |
| vda | 4345 | 1954 | 4342 | 4048 | 4214 | 4293 | 4437 | 4344 |
| too_large | 7096 | 14507 | 7097 | 7097 | 7091 | 7090 | 7095 | 7095 |
| term1 | 580 | 1161 | 590 | 586 | 584 | 584 | 579 | 579 |
| pair | 67685 | 108998 | 68085 | 68085 | 68006 | 68007 | 67983 | 68012 |
| my_adder | 327677 | 589831 | 196614 | 196613 | 196613 | 196613 | 196608 | 196608 |
| mux | 131071 | 217 | 131072 | 131072 | 131072 | 131072 | 131072 | 131070 |
| k2 | 28336 | 5986 | 27474 | 27460 | 26361 | 27518 | 28137 | 28335 |
| i9 | 2278 | 8754 | 3751 | 2277 | 2277 | 2277 | 2277 | 2277 |
| i8 | 4366 | 14750 | 5208 | 4466 | 4433 | 4385 | 4365 | 4365 |
| i7 | 505 | 1000 | 633 | 504 | 632 | 578 | 504 | 504 |
| i5 | 312 | 763 | 523 | 460 | 523 | 451 | 329 | 317 |
| i4 | 421 | 1095 | 430 | 430 | 430 | 430 | 428 | 420 |
| i2 | 335 | 317 | 336 | 336 | 334 | 334 | 334 | 334 |
| frg2 | 6471 | 5031 | 6235 | 6326 | 6348 | 6344 | 6354 | 6465 |
| frg1 | 204 | 383 | 208 | 206 | 206 | 204 | 203 | 203 |
| example2 | 469 | 644 | 457 | 473 | 456 | 456 | 454 | 454 |
| count | 234 | 294 | 226 | 235 | 226 | 226 | 227 | 225 |
| cm150a | 131071 | 220 | 131072 | 131072 | 131072 | 131072 | 131070 | 131070 |
| bw6x6 | 830 | 3535 | 1733 | 1732 | 1714 | 1705 | 1693 | 1675 |
| b9 | 178 | 318 | 204 | 190 | 198 | 196 | 182 | 179 |
| apex7 | 1660 | 1221 | 1736 | 1569 | 1570 | 1556 | 1536 | 1659 |
| apex1 | 28336 | 8901 | 38460 | 37389 | 26238 | 26314 | 28324 | 28337 |
| alu4 | 1182 | 2141 | 1491 | 1641 | 1269 | 1265 | 1253 | 1243 |
| alu32r | 189266 | 18629 | 185397 | 185395 | 185394 | 185380 | 185361 | 188583 |
| alu32 | 12194 | 959 | 8161 | 8161 | 8303 | 8686 | 10709 | 12193 |
| alu2 | 231 | 420 | 297 | 240 | 237 | 234 | 232 | 232 |
| adsb32r | 528 | 897 | 698 | 688 | 688 | 687 | 685 | 623 |
| adder16 | 327812 | 606303 | 458850 | 458850 | 262310 | 262308 | 262293 | 262293 |
| C499 | 45922 | 13699 | 7093 | 7093 | 7029 | 7029 | 7029 | 7029 |
| C432 | 1733 | 4913 | 1342 | 1342 | 1470 | 1736 | 1732 | 1732 |
| C1908 | 36007 | 37800 | 41792 | 40971 | 35869 | 36013 | 36009 | 36007 |
| C1355 | 45922 | 14162 | 47515 | 45921 | 45921 | 45921 | 45921 | 45921 |
| bigkey | 6170 | 7970 | 5518 | 5518 | 6188 | 6188 | 6176 | 6172 |
| $\Sigma$ | 4.468.873 | 4.687.023 | 3.216.892 | 3.215.694 | **2.997.931** | 2.999.501 | 3.009.641 | 3.015.341 |
|  | 100% | 104.9% | 72.0% | 72.0% | **67.1%** | 67.2% | 67.3% | 67.5% |

**Table 3**: Heuristic for $\oplus$-Node Placement – (MAX/nDE) – Single Circuits.