

Improved Algorithms for Low Power Multiplexor Decomposition¹

Shaofa Yang, Hon Wai Leong

Department of Computer Science, National University of Singapore,
3 Science Drive 2, Singapore 117543

Email: yangsf, leonghw@comp.nus.edu.sg

Abstract: It has been estimated that multiplexors (MUXes) make up a major portion of the circuitry in a typical chip. Therefore, to reduce power consumption of a chip, it is important to consider the design of MUXes that consumes less power. This is called the *low power MUX decomposition* problem and has been studied in [6]. This paper improves on the results of [6] in two ways: (a) we propose a method to speed up the algorithms in [6], and (b) we propose a post-optimization procedure to further reduce the overall power dissipation of decompositions obtained by *any* MUX decomposition algorithm. Using this post-optimization procedure, we have been able to further reduce the power dissipation results of [6].

Keywords: multiplexor, power minimization, logic decomposition, design automation, algorithm

1 Introduction

This paper studies the *low power multiplexor (MUX) decomposition (LPMD)* problem first introduced by Narayanan *et al.* [6]. Research on low power circuit design is driven by the widespread use of portable electronic devices such as laptop computers and personal digital assistants which have very limited battery life, and by the need to reduce heat dissipation (thus increasing reliability) in large complex circuits. It has been estimated that multiplexors make up a major portion of the circuitry in a typical chip [6, 10]. In particular, multiplexors could account for as much as 46.70% of the overall power consumption in a control-flow intensive circuit [9]. Thus, it is important to reduce power dissipation of the *many* MUXes in circuits. Further, *fast* algorithms are needed for fast design space exploration of large circuits, which could contain thousands of MUXes. The results of [6] have since been used in several related contexts [4, 7].

The LPMD problem is to transform an n -to-1 MUX into a logically equivalent *MUX tree of 2-to-1 MUXes* that has the *minimum* overall power dissipation. The *overall power dissipation* of an MUX tree is the sum of power dissipation of all 2-to-1 MUXes in the tree. For example, Fig. 1 shows two different decompositions of an 8-to-1 MUX. Note that the two decompositions are *logically* equivalent but they may have different overall power dissipation.

Low power logic decomposition of simple logic gates such as AND and OR gates has been widely studied in [5, 8, 11]. Thakur *et al.* [10] studied MUX decomposition that minimizes delay but did not consider power dissipation. The LPMD problem was first investigated by Narayanan *et al.* [6]. Kim *et al.* [3] also studied the LPMD problem, but with a different power estimation model from [6]. In this paper, we consider only the power model of [6]. We

¹A preliminary version appeared in *Proc. of 10th Int. Symp. on Integrated Circuits, Devices and Systems, 2004*.

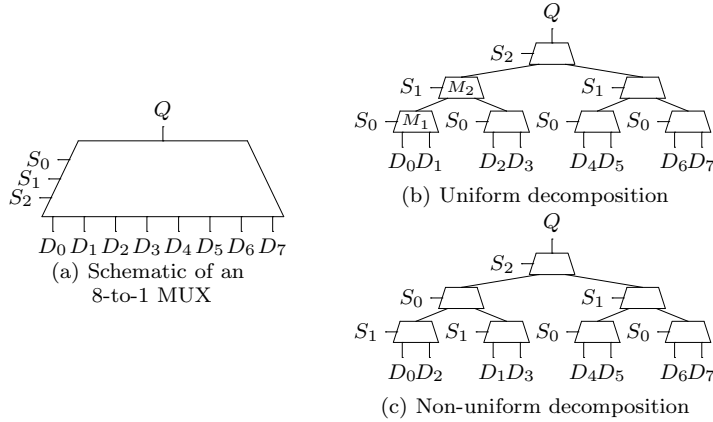


Figure 1: Two different decompositions of an 8-to-1 MUX

mention also the work of Khouri *et al.* [2], which considers the problem of multiplexor tree restructuring targeting power reduction at the *RT level* in control-flow intensive circuits.

This paper is organized as follows. In Section 2, we present the LPMD problem formulation following closely [6] and review the MUX decomposition algorithms proposed therein. In Section 3, we present a fast algorithm for pre-computing a *MUX ON-probability table* and show how this is used to speed up the MUX decomposition algorithms in [6]. In Section 4, we give the *Compatibility Theorem* for generalized MUX trees, that forms the basis of *any* non-uniform bottom-up strategy for solving the LPMD problem. Based on the Compatibility Theorem, we propose a *post-optimization* procedure for improving the solutions obtained by any existing algorithm. In Section 5, we present experimental results on randomly generated datasets as well as on a “simulated” MUX in a MIPS processor. The results show that our speed-up method and post-optimization procedure are both effective. In the concluding section, we discuss a number of future directions.

2 Problem Formulation and Existing Algorithms

We adopt the LPMD problem formulation of [6]. In the LPMD problem, we are given an n -to-1 MUX \mathcal{M} with n data signals D_0, D_1, \dots, D_{n-1} and k selection signals S_0, S_1, \dots, S_{k-1} where $n = 2^k$. An example of an 8-to-1 MUX is shown in Fig. 1(a) with data signals D_0, D_1, \dots, D_7 and selection signals S_0, S_1, S_2 . By convention, D_j will be “selected” to be the output Q by a combination $(S_2 S_1 S_0)$ of selection signal values, where j is the decimal equivalent of $(S_2 S_1 S_0)$. The combination $(S_2 S_1 S_0)$ is called the *encoding* of D_j . For example, the encoding of D_6 is (110), that is, D_6 will be “selected” when $S_2 = 1, S_1 = 1, S_0 = 0$.

The inputs to the LPMD problem are the *ON-probabilities* and the *occurrence probabilities* of all data signals. The *ON-probability* d_j is the probability that the data signal D_j is high. The *occurrence probability* $P(D_j)$ is the probability that the data signal D_j is selected to be the output of \mathcal{M} . The task is to decompose the n -to-1 MUX \mathcal{M} into a MUX tree T consisting of 2-to-1 MUXes so as to minimize the *overall power dissipation*, that is, the sum of power dissipation over all 2-to-1 MUXes in T .

A MUX decomposition is *balanced* if all paths from the root MUX to the data signals are of equal length. The two MUX decompositions in Fig. 1 are both balanced. This paper

considers only balanced decompositions. A *uniform* MUX decomposition is one in which all MUXes at the same level *use the same* selection signal. A *non-uniform* decomposition is one in which MUXes at the same level *can use different* selection signals. This is illustrated in Fig. 1.

2.1 Power Model for MUXes

While a CMOS 2-to-1 MUX M is a more complex circuit compared to a simple AND or OR gate, Narayanan *et al.* [6] showed that the formula for the power dissipation of M is similar to that of a simple AND or OR gate. Namely, the power dissipation of a CMOS 2-to-1 MUX M is given by $2 \cdot P_M \cdot (1 - P_M)$, where P_M is the ON-probability of the *output* or *fan-out signal* of M (or simply the *ON-probability of M*). That is, for any 2-to-1 MUX M , the power dissipation is *completely determined* by P_M . It follows that obtaining the overall power dissipation of a MUX tree T amounts to computing the ON-probabilities of all MUXes in T [6].

2.2 Computation of MUX ON-Probabilities

For a standalone 2-to-1 MUX \mathcal{M} with data signals D_0 and D_1 , the ON-probability of the fan-out signal of \mathcal{M} is given by $P_{\mathcal{M}} = P(D_0)d_0 + P(D_1)d_1$. For the general n -to-1 MUX, the computation is more involved. In a MUX tree T , a MUX M is called a *leaf MUX* if both of its fan-in signals are data signals, otherwise it is called an *internal MUX* [6]. Consider the leaf MUX M_1 and the internal MUX M_2 in Fig. 1(b). For leaf MUX M_1 , the ON-probability is given by $P_{M_1} = \Pr(S_0 = 0) \cdot d_0 + \Pr(S_0 = 1) \cdot d_1$, where $\Pr(S_0 = 0)$ is the probability that the selection signal S_0 is low and $\Pr(S_0 = 1)$ is the probability that the selection signal S_0 is high. These selection signal probabilities can be derived from the occurrence probabilities of the data signals. For example, $\Pr(S_0 = 0) = P(D_0) + P(D_2) + P(D_4) + P(D_6)$.

Calculating the ON-probabilities of internal MUXes is more complicated. For example, the ON-probability of the internal MUX M_2 (in Fig. 1(b)) is given by $P_{M_2} = \Pr(S_1S_0 = 00) \cdot d_0 + \Pr(S_1S_0 = 01) \cdot d_1 + \Pr(S_1S_0 = 10) \cdot d_2 + \Pr(S_1S_0 = 11) \cdot d_3$. Here $\Pr(S_1S_0 = 00)$ is the probability that S_1 is low and S_0 is low, and is given by $P(D_0) + P(D_4)$. Similarly, $\Pr(S_1S_0 = 01)$ is given by $P(D_1) + P(D_5)$, and so on. Observe that the ON-probability of the internal MUX M_2 depends on d_0, d_1, d_2, d_3 , namely, the ON-probabilities of *all* the data signals in its fan-in tree.

For the general case, let $E(D_j)$ be the encoding of D_j . Further, let $E_r(D_j)$ denote the bit of $E(D_j)$ corresponding to selector S_r . For any (leaf or internal) MUX M in a MUX tree T , the ON-probability of M is given by ([6])

$$P_M = \sum_{D_j \in D(M)} d_j \cdot \Pr(SP(D_j, M)) , \quad (1)$$

where $D(M)$ is the set of data signals in the fan-in tree of M , and $\Pr(SP(D_j, M))$ is the probability that D_j is “selected” to be the fan-out of M . Let $S(M)$ be the set of selection signals in the fan-in tree of M . Then $\Pr(SP(D_j, M))$ is the sum of $P(D_i)$ over all D_i where $E_r(D_i) = E_r(D_j)$ for every $r \in S(M)$. (We refer the reader to [6] for explanations of formula (1) given above.) Finally, we recall an important result from [6].

Theorem 1 (Independence Theorem). *In an MUX decomposition of an n -to-1 MUX, the ON-probability of any MUX M is independent of the order of selection signals in the fan-in tree of M .*

As it turns out, the Independence Theorem is the key to our improved algorithms with faster running times. This is described in Section 3.

2.3 MUX Decomposition Algorithms

We now briefly review the MUX decomposition algorithms in [6]. Three heuristic algorithms were presented for solving the LPMD problem: BOTTOM-UP, TOP-DOWN and HYBRID. BOTTOM-UP is a greedy bottom-up (level-by-level from leaf to root) strategy that produces only uniform decompositions. At each level r , it chooses the selection signal which minimizes the total power dissipation for all the MUXes at level r . TOP-DOWN constructs a non-uniform decomposition from root to leaves. It first assigns the selection signal to the root MUX M that minimizes the total power dissipation of the two children MUXes of M . After that, TOP-DOWN is recursively applied on the two sub-trees of the root MUX. HYBRID constructs the tree in the same order as TOP-DOWN. However, for each MUX M , it assigns to M the same selection signal as BOTTOM-UP decomposition (of the sub-tree rooted at M) would do. An exhaustive search algorithm called BRANCH-AND-BOUND, which finds an optimal decomposition, was also presented. Finally, we remark that all the algorithms described have been extended to handle incomplete n -to-1 MUXes by setting both the ON-probability and occurrence probability of every “don’t-care” data signal to be 0. We refer to [6] for details of these algorithms.

3 MUX On-Probability Table for Speeding up Existing Algorithms

The decomposition algorithms in [6] compute the ON-probability of each internal MUX using formula (1), which is expensive since it depends on all data signals in its fan-in tree [6]. This leads to inefficiencies in these algorithms.

In this paper, we note that the Independence Theorem implies that the ON-probability of a given internal MUX depends *only* on the data signals in its fan-in tree and remains unchanged for *all possible decompositions* of its fan-in tree. Therefore, its ON-probability can be pre-computed, independent of its eventual decomposition. This implies that we can *pre-compute* a MUX ON-probability table (*MOT*), a table of ON-probabilities for *all possible* (2-to-1) MUXes. We then use the *MOT* for fast *table lookup* of ON-probabilities and significantly speed up the existing algorithms for LPMD presented in [6].

3.1 Encoding of MUX Fan-out Signals

To this end, we first give a *ternary encoding* for the fan-out signals of all MUXes in a MUX tree. For a leaf MUX M with data signals D_i and D_j and selector S_r , we encode the fan-out signal of M by $E(M) = (e_{k-1} \dots e_1 e_0)$ where $e_r = \text{“x”}$ and $e_h = E_h(D_i) = E_h(D_j)$ for every $h \neq r$. We call $E(M)$ the (*fan-out*) *encoding* of M . This is illustrated in Fig. 2(a) where $E(M_a) = (0x100)$. Similarly, we extend this to internal MUXes as shown in Fig. 2(b).

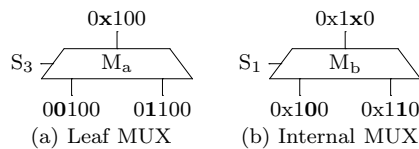


Figure 2: Encoding 2-to-1 MUXes

It follows from the Independence Theorem that any two MUXes (in different decompositions of the same n -to-1 MUX) have equal ON-probabilities (and thus power) if their encodings are identical. Since we are only interested in power dissipation of MUXes, we shall, from now on, refer to a MUX M by its fan-out encoding $E(M)$. With this extension of encodings to MUX fan-out signals, we have a uniform framework to talk about all the data and MUX fan-out signals in a MUX tree.

In this extension, we go from a binary encoding of the data signals to a ternary encoding $(e_{k-1} \dots e_0)$ where each e_j can take value 0, 1, x (with “x” being regarded as “2”). Therefore, there are altogether 3^k possible signal encodings. These consist of the given 2^k data signal encodings and the remaining $(3^k - 2^k)$ MUX encodings.

Let $P_{on}(e_{k-1} \dots e_0)$ denote the ON-probability of a MUX (or data signal) whose encoding is $(e_{k-1} \dots e_0)$. We shall store all the 3^k ON-probabilities in a table, called the *MUX ON-probability table (MOT)*. And we index the MOT by the ternary MUX (and data signal) encodings. More precisely, $MOT[J] = P_{on}(e_{k-1} \dots e_0)$ where J is the decimal equivalent of $(e_{k-1} \dots e_0)$. We show that this MOT can be computed efficiently optimally in $O(3^k)$ time – constant time per entry.

3.2 Fast Pre-computation of the MOT

Let M, M_0, M_1 be MUXes of encodings $E(M) = (e_{k-1} \dots e_{r+1} \mathbf{x} e_{r-1} \dots e_0)$, $E(M_0) = (e_{k-1} \dots e_{r+1} \mathbf{0} e_{r-1} \dots e_0)$, and $E(M_1) = (e_{k-1} \dots e_{r+1} \mathbf{1} e_{r-1} \dots e_0)$. In other words, if we were to assign S_r to be the selector of M , then the two fan-in signals of M are precisely M_0, M_1 . It is easy to show (see [6]) that

$$P_{on}(e_{k-1} \dots \mathbf{x} e_{r-1} \dots e_0) = \Pr(S_r = 0) \cdot P_{on}(e_{k-1} \dots \mathbf{0} e_{r-1} \dots e_0) + \Pr(S_r = 1) \cdot P_{on}(e_{k-1} \dots \mathbf{1} e_{r-1} \dots e_0). \quad (2)$$

For the MUX shown in Fig. 2(b), we have $P_{on}(0x1x0) = \Pr(S_1 = 0) \cdot P_{on}(0x100) + \Pr(S_1 = 1) \cdot P_{on}(0x110)$. We emphasize that, by the Independence Theorem, the above formula holds even if the selector assigned to M is *not* S_r . Consequently, we can pre-compute the MOT independent of the MUX decomposition process.

With Eq. (2), we can pre-compute the MOT in time $O(3^k)$ by *suitably scheduling* our computations. We refer to [12] for the detailed algorithm. Here we illustrate the computation for an 8-to-1 MUX ($n = 8, k = 3$) in Fig. 3. We first pre-compute the weights $\Pr(S_i = 0), \Pr(S_i = 1), i = 0, 1, 2$, as explained in Section 2.2.

We start with the left-most sub-table $P_{on}(\ast)$ of ON-probabilities of the 8 data signals, namely, d_0, \dots, d_7 . To get the next sub-table, $P_{on}(\mathbf{S}_2)$, we first partition $P_{on}(\ast)$ along S_2 into 4 pairs $(P_{on}(000), P_{on}(100)), (P_{on}(001), P_{on}(101)), (P_{on}(010), P_{on}(110)),$ and $(P_{on}(011), P_{on}(111))$. Then the four elements of sub-table $P_{on}(\mathbf{S}_2)$ are obtained from the weighted sum of each pair with weights $\Pr(S_2 = 0), \Pr(S_2 = 1)$, following Eq. (2). Similarly, partitioning



Figure 3: Pre-computation of the MOT for an 8-to-1 MUX

$P_{on}(\ast)$ along S_1 and summing up each pair with weights $\Pr(S_1 = 0), \Pr(S_1 = 1)$ yields sub-table $P_{on}(\mathbf{S}_1)$. In a similar way, we compute sub-table $P_{on}(\mathbf{S}_0)$.

Next, we partition sub-table $P_{on}(S_2)$ along S_1, S_0 respectively to obtain sub-tables $P_{on}(S_2\mathbf{S}_1), P_{on}(S_2\mathbf{S}_0)$. Similarly, partitioning $P_{on}(S_1)$ along S_0 yields sub-table $P_{on}(S_1\mathbf{S}_0)$. Finally, we partition sub-table $P_{on}(S_2\mathbf{S}_1)$ along S_0 to get the last sub-table $P_{on}(S_2\mathbf{S}_1\mathbf{S}_0)$. This completes the computation of all the 27 entries of the MOT. Note that to compute each entry it takes exactly two multiplications and one addition – namely, $O(1)$ time.

In general, for an n -to-1 MUX with k selection signals (where $n = 2^k$), we start with the initial sub-table $P_{on}(\ast)$ containing the ON-probabilities of the n given data signals, namely d_0, \dots, d_{n-1} . From the sub-table $P_{on}(\ast)$, we partition along S_{k-1}, \dots, S_1, S_0 to get the sub-tables $P_{on}(\mathbf{S}_{k-1}), \dots, P_{on}(\mathbf{S}_1), P_{on}(\mathbf{S}_0)$, respectively. Then, *recursively* for each sub-table $P_{on}(S_{j_1}S_{j_2}\dots S_{j_r})$ where $j_1 > j_2 > \dots > j_r > 0$, we partition it along $S_{j_r-1}, S_{j_r-2}, \dots, S_0$ to get sub-tables $P_{on}(S_{j_1}S_{j_2}\dots S_{j_r}\mathbf{S}_{j_r-1}), P_{on}(S_{j_1}S_{j_2}\dots S_{j_r}\mathbf{S}_{j_r-2}), \dots, P_{on}(S_{j_1}S_{j_2}\dots S_{j_r}\mathbf{S}_0)$, respectively. As each computation involves only two multiplications and one addition, this procedure computes the MOT in time $O(3^k)$, which is obviously optimal.

3.3 Modified Algorithms

We then use the *MOT* to speed up the existing decomposition algorithms in [6] (BOTTOM-UP, TOP-DOWN, HYBRID, and BRANCH-AND-BOUND) as follows: We modify each algorithm to first compute the *MOT*. Then, for each MUX under consideration, modify it to lookup the ON-probability from the *MOT* instead of computing it on the fly. The initial computation of the *MOT* induces an overhead, but this is more than compensated for by the savings obtained during the execution of the decomposition algorithms.

4 A New Post-Optimization Procedure

Our second contribution is a post-optimization procedure that can further reduce the power dissipation of MUX decompositions given by any decomposition algorithm. In this paper, we apply this to the heuristic algorithms in [6]. This post-optimization procedure is derived from the following *Compatibility Theorem*.

4.1 Compatibility Theorem

One drawback of the decomposition algorithm BOTTOM-UP is that it considers only *uniform* decompositions. This stems from the fact that considering non-uniform decompositions in a bottom-up scheme is complicated. Given a set of n data signals, we can combine them into $\frac{n}{2}$ 2-to-1 MUXes. However, not all such sets of $\frac{n}{2}$ MUXes are *compatible*, i.e. can be eventually combined into a MUX tree. We prove the *Compatibility Theorem*, which enables us to efficiently check whether a set of MUXes are compatible.

First we generalize the notion of MUX trees. A *generalized MUX tree* is a balanced tree of 2-to-1 MUXes whose “leaf signals” can be either data signals or MUX fan-out signals. For example, Fig. 4 shows a generalized MUX tree with root MUX 01xxx and leaf signals $\{0100x, 0110x, 01x10, 01x11\}$. A set of MUX fan-out signals \mathcal{F} is *compatible* if there exists a generalized MUX tree whose set of leaf signals is equal to \mathcal{F} .

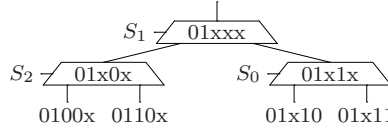


Figure 4: A generalized MUX tree

Let \mathcal{F} be a set of MUX fan-out signals and S_r a selector. We say S_r is a *candidate root selector* of \mathcal{F} if there are exactly $\frac{|\mathcal{F}|}{2}$ members M in \mathcal{F} with $E_r(M) = 0$ and exactly $\frac{|\mathcal{F}|}{2}$ members M in \mathcal{F} with $E_r(M) = 1$ (and no members M in \mathcal{F} with $E_r(M) = x$). Finally, we say two MUX trees are *equivalent* iff they have identical sets of leaf signals. We have the following:

Theorem 2 (Compatibility Theorem). *Let \mathcal{F} be a set of MUX fan-out signals and S_r an (arbitrary) candidate root selector for \mathcal{F} . Then \mathcal{F} is compatible iff both \mathcal{F}_0 and \mathcal{F}_1 are compatible, where $\mathcal{F}_i = \{M \in \mathcal{F} : E_r(M) = i\}$ for $i = 0, 1$.*

Proof. The “if” part is immediate. Now consider the “only if” part. Suppose T is a MUX tree and \mathcal{F} is its set of leaf signals. Define the *level of S_r in tree T* , denoted by $level(S_r)$, to be the maximum level of a MUX in T which uses S_r as its selector. (The root MUX is defined to be at level 0.) If $level(S_r) = 0$, that is, S_r is the selector of the root MUX of T , then it follows that \mathcal{F}_0 and \mathcal{F}_1 are both compatible.

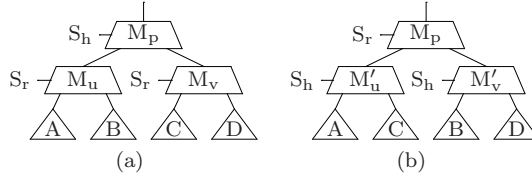


Figure 5: Exchanging MUX selectors

Note that S_r must appear in *every* path of T . Let $level(S_r) = \ell$. Consider a MUX M_u at level ℓ of which S_r is the selector, as shown in Fig. 5(a). It follows that M_v , the “sibling” MUX of M_u , must also use S_r as its selector. Hence, we can exchange the selectors of M_u, M_v with that of their parent MUX M_p and obtain a generalized MUX tree (shown in Fig. 5(b))

which is equivalent to T . Repeat this for every MUX at level ℓ for which S_r is the selector, we obtain a generalized MUX tree T' in which the level of S_r is $\ell - 1$. And T' is equivalent to T .

It follows by induction that we can always transform the tree T into a generalized MUX tree whose root MUX uses S_r as the selector. This establishes the Compatibility Theorem. \square

The Compatibility Theorem enables us to quickly check whether a set of MUXes \mathcal{F} is compatible. We just pick *any* candidate root selector S_r , partition \mathcal{F} into \mathcal{F}_0 and \mathcal{F}_1 along S_r , and recursively check the compatibility of \mathcal{F}_0 and \mathcal{F}_1 . (This procedure can then be used for deriving non-uniform bottom-up MUX decomposition algorithms, though we have not exploited this in the present paper.)

4.2 Post-Optimization Procedure

Our post-optimization procedure is based on the selector exchange operation used in the proof of the Compatibility Theorem. Suppose T is a MUX tree for an n -to-1 MUX \mathcal{M} . Let M_p be an internal MUX of T whose immediate fan-in MUXes are M_u, M_v and suppose that M_u, M_v use the same selector S_r (as shown in Fig. 5(a)). Then, recall from the proof of the Compatibility Theorem, that we can *exchange* the selector of M_p with those of M_u and M_v to obtain another MUX decomposition T' (shown in Fig. 5(b)) that is logically equivalent to T . Furthermore, by the Independence Theorem, the ON-probabilities (and hence, the power dissipation) of all the MUXes in T *remain unchanged* in T' , except for those of M_u, M_v . Therefore, we can quickly compute the change in power dissipation from T to T' , and perform the corresponding exchange operation when the overall power dissipation decreases.

This suggests a general post-optimization schema as follows: Given a MUX tree, we iteratively exchange MUX selectors whenever the overall power dissipation is reduced. We stop when we cannot reduce power anymore via an exchange operation. We implement two variations of this schema: LEVEL-POST and GREEDY-POST.

LEVEL-POST considers one level of MUXes at a time. In each pass, we traverse the MUX tree level by level from leaf to root. We stop after the pass which makes no exchange. The rationale behind LEVEL-POST is that exchange decisions are based on an entire level of MUXes and there are more MUXes near the leaf level (than near the root level).

GREEDY-POST considers one MUX at a time. It iteratively perform the exchange operation that gives the largest power reduction.

5 Experimental Results

We have implemented the $O(3^k)$ algorithm for computing the MOT and used it in our modified version of the algorithms from [6], called BOTTOM-UP', TOP-DOWN', HYBRID', and BRANCH-AND-BOUND'², respectively. In these algorithms, there are two steps, the first pre-computes the MOT and the second modifies the corresponding algorithm in [6] to *lookup* the ON-probabilities from the MOT . We have also implemented the two post-optimization algorithms LEVEL-POST and GREEDY-POST. Our implementations are in C++ running on one node of a 32-node Fujitsu AP3000 system. To measure the improvements of speed-up, we have also obtained the software package (which is in C++ as well) from [6], recompiled and

²In this paper, we refer only to the version which finds an optimal *non-uniform* decomposition.

run it on our machine. We use two types of datasets in our evaluation: (a) randomly generated datasets similar to those used in [6], and (b) “simulated” MUXes for a MIPS instruction class of the SPECfp2000 benchmark.

5.1 Evaluation using Randomly Generated Datasets

The datasets are generated in the same manner as in [6]. We allow the occurrence probabilities of data signals, $P(D_j)$'s, to vary uniformly over $\frac{1}{n}(1 \mp \alpha)$ ($0 \leq \alpha < 1$), and the ON-probabilities of data signals, d_j 's, to vary uniformly over $0.5(1 \mp \beta)$ ($0 \leq \beta < 1$). For incomplete MUXes, we use an additional parameter γ ($0 \leq \gamma < 0.5$) to control the percentage of “don't-care” data signals.

Table 1: Our Speed-up of the Various MUX Decomposition Algorithms

n	BOTTOM-UP	BOTTOM-UP'			Speed-up Δ/Δ'
	Δ (s)	Δ_1 (s)	Δ_2 (s)	Δ' (s)	
16	0.03	0.01	0.01	<i>0.02</i>	1.5
32	0.13	0.02	0.03	<i>0.05</i>	2.6
64	0.56	0.08	0.08	<i>0.16</i>	3.5

n	TOP-DOWN	TOP-DOWN'			Speed-up Δ/Δ'
	Δ (s)	Δ_1 (s)	Δ_2 (s)	Δ' (s)	
16	0.01	0.01	0.00	<i>0.01</i>	1.0
32	0.06	0.02	0.01	<i>0.03</i>	2.0
64	0.28	0.08	0.02	<i>0.10</i>	2.8

n	HYBRID	HYBRID'			Speed-up Δ/Δ'
	Δ (s)	Δ_1 (s)	Δ_2 (s)	Δ' (s)	
16	0.07	0.01	0.02	<i>0.03</i>	2.3
32	0.33	0.02	0.07	<i>0.09</i>	3.7
64	1.61	0.08	0.21	<i>0.29</i>	5.6

n	BRANCH-AND-BOUND	BRANCH-AND-BOUND'			Speed-up Δ/Δ'
	Δ (s)	Δ_1 (s)	Δ_2 (s)	Δ' (s)	
16	0.11	0.01	0.03	<i>0.04</i>	2.8
32	1.45	0.02	0.29	<i>0.31</i>	4.7
64	27.17	0.08	3.73	<i>3.81</i>	7.1

Speed-up Results: Table 1 shows the improvements in running times (in seconds). In the table, Δ and Δ' are the running times of the original algorithms and our version of the algorithms, respectively. For our versions, Δ_1 is the time taken to pre-compute the MOT, while Δ_2 is the time taken to perform decomposition ($\Delta' = \Delta_1 + \Delta_2$). The speed-up obtained are shown in **bold** in the last column. For each n ranging over $\{16, 32, 64\}$ ³, we ran both programs over 75 sets of data, where α ranges over $\{0.1, 0.2, 0.3, 0.45, 0.5\}$, β over $\{0.5, 0.8, 0.9\}$, and γ over $\{0, 0.2, 0.3, 0.4, 0.49\}$, and obtain the average running time.

As expected, we obtain a speed-up in running time for all algorithms. The time taken to pre-compute the MOT, Δ_1 , is small for all n . The actual speed-up obtained depends on the number of MUX ON-probability computations that are carried out during the algorithm. TOP-DOWN performs fewest ON-probability computations and so its speed-up is not very

³The running times for $n=8$ (8-to-1 MUXes) are not included as they are too small.

high (up to **2.0** for $n=32$). For HYBRID and BRANCH-AND-BOUND, we obtain a speed-up of **3.7** and **4.7** for $n=32$, respectively.

Though the running times for decomposition of a single n -to-1 MUX are small, we point out again that a typical circuit could contain thousands of MUXes. Further, when exploiting the (usually huge) design space of a circuit, it is important to be able to quickly estimate the minimum power dissipation of design alternatives.

Table 2: Power Reduction obtained by LEVEL-POST and GREEDY-POST.

(α, β, γ)	n	BOTTOM-UP'			TOP-DOWN'			HYBRID'		
		Orig	Level	Greedy	Orig	Level	Greedy	Orig	Level	Greedy
(.5, .8, 0)	8	0.5	*	*	*	*	*	*	*	*
	16	2.6	*	*	*	*	*	*	*	*
	32	3.6	1.9	1.9	1.2	1.2	1.2	1.2	1.2	1.2
	64	3.9	1.8	1.8	2.9	2.9	2.9	1.9	1.9	1.9
(.2, .5, .2)	8	*	*	*	0.6	*	*	*	*	*
	16	1.1	0.9	0.9	0.1	*	*	1.0	1.0	1.0
	32	1.5	0.4	0.4	3.9	3.7	1.9	0.1	*	*
	64	3.3	2.0	2.0	3.1	0.3	0.3	1.2	1.2	1.2
(.3, .9, .3)	8	3.4	3.4	3.4	5.3	0.9	0.9	3.4	3.4	3.4
	16	11.8	6.2	*	9.6	7.2	7.2	4.7	*	*
	32	13.8	6.3	6.3	13.3	6.0	6.0	4.3	4.3	4.3
	64	8.7	2.4	1.2	7.8	3.7	2.3	2.3	2.1	2.1
(.5, .9, .49)	8	18.3	18.3	18.3	19.7	1.3	1.3	18.3	18.3	18.3
	16	12.2	8.0	3.8	6.4	6.4	6.4	1.2	*	*
	32	3.0	1.4	1.4	11.8	10.1	4.4	3.6	1.8	1.8
	64	15.4	8.5	8.5	18.6	7.8	7.8	11.3	10.7	10.7

Post-Optimization Results: Next we apply the post-optimization methods LEVEL-POST and GREEDY-POST to obtain further power reduction in the solutions obtained by the algorithms BOTTOM-UP', TOP-DOWN', and HYBRID'. Table 2 shows a sampling of the power reduction obtained. Each entry in the table shows the power dissipation in the form of λ , the *percentage above the optimal power dissipation*. We use algorithm BRANCH-AND-BOUND' to obtain the optimal power dissipation. For instance, an entry of 2.6 ($\lambda=2.6$) means that the power of the decomposition obtained is 102.6% that of an optimal decomposition. An entry “*” indicates that an optimal decomposition ($\lambda=0$) is obtained. For each heuristic algorithm (BOTTOM-UP', TOP-DOWN', and HYBRID'), we show the value of λ for the original algorithm, and that after applying LEVEL-POST and GREEDY-POST to the original solution.

In general, HYBRID' generates *near* optimal decompositions, thus only small improvements can be achieved by applying post-optimization. For TOP-DOWN' (which usually performs worse than HYBRID') *better improvements* are obtained by our post-optimization procedures. BOTTOM-UP' generally performs worst since it only considers uniform decompositions⁴. However, it is *significant* to observe that by applying our post-optimization after BOTTOM-UP', we are able to get results that are *as good as* those produced by HYBRID'. Overall, the reduction in power dissipation increases with the size of the MUXes. Our more extensive results (not shown here, but see [12] for details) also indicate that it increases with the percentage of “don't-care” data signals. LEVEL-POST and GREEDY-POST are equally effective in most cases and GREEDY-POST is slightly better when they differ.

⁴The values of λ for BOTTOM-UP' are calculated based on optimal *non-uniform* decompositions.

5.2 Evaluation using a Simulated MUX Design

While real MUX designs are available, we were not able to find any that comes with known ON-probabilities d_j and occurrence probabilities $P(D_j)$ of the data signals. Instead, we use the following method to obtain a “simulated” MUX in a MIPS processor. We first obtain the frequency of usage of a set of MIPS instruction classes from the well-known textbook by Hennessy and Patterson [1] (Figure 2.33). The table shows the relative frequency of a set of 27 MIPS instruction classes for a set of SPECfp2000 benchmark programs.

To simulate the corresponding MUX used for the output register of the MIPS processor, we use a 32-to-1 MUX (with 5 “don’t-care” data signals) where the instruction classes are the input data signals D_0, D_1, \dots, D_{31} to the MUX and their relative frequencies correspond to their occurrence probabilities $P(D_j)$'s. We also assume the ON-probabilities d_j are equal to the occurrence probabilities – namely, that the data signal of each instruction class is high exactly when it is selected. We assume that the instructions classes are encoded with 5 selection signals $(S_4S_3S_2S_1S_0)$ in the order listed in the table. Finally, for comparison purposes, we start with the *default* decomposition of this 32-to-1 MUX in which S_4 is assigned to the root MUX (level 0), S_3 to all MUXes at level 1, \dots , and S_0 to all leaf MUXes.

Table 3: MUX Decomposition Results for a Simulated 32-to-1 MUX in a MIPS Processor

benchmk	default	BOTTOM-UP'			TOP-DOWN'			HYBRID'		
		Orig	Level	Greedy	Orig	Level	Greedy	Orig	Level	Greedy
applu	16.6	4.0	0.4	0.4	0.4	0.4	0.4	*	*	*
art	14.3	6.1	4.0	2.7	2.9	2.9	2.9	1.8	1.8	1.8
equake	15.1	1.9	1.7	1.5	2.8	2.6	2.6	0.2	0.2	0.2
lucas	10.7	4.0	0.1	0.1	0.9	*	*	0.3	*	*
swim	16.1	3.1	0.3	0.3	2.7	0.3	0.3	0.8	0.8	0.8
FPavg	16.0	4.9	0.7	0.7	1.3	1.3	1.3	0.1	0.1	0.1

We then ran this simulated MUX design through the various MUX decomposition algorithms, followed by the post-optimization procedures. The results obtained are shown in Table 3. Table 3 shows firstly, that *all* the MUX decomposition algorithms are effective in reducing the power dissipation over the *default* decomposition. Secondly, the general trends in Table 2 and Table 3 are very similar. This suggests that the performance of our algorithms in real MUXes will be similar to those for the randomly generated datasets (which we have extensively evaluated).

6 Conclusions

This paper presents two improvements to the solutions in Narayanan *et al.* [6] for the low power MUX decomposition (LPMD) problem: a speed-up method for existing LPMD algorithms in [6] and a post-optimization procedure to further reduce power dissipation. Our experimental results indicate that we can achieve good speed-ups (3.7 times over the HYBRID algorithm for 32-to-1 MUXes). Our experiments also show that the post-optimization procedure is effective. It is significant that it can take the result of a uniform decomposition (from BOTTOM-UP') and improve it to be roughly equivalent to that obtained by the best heuristic algorithm – HYBRID'.

There are a number of directions for future works. With the pre-computed MOT, one can afford to do more algorithmic exploration and may thus obtain better heuristic algorithms

for the LPMD problem. It is certainly interesting to exploit the Compatibility Theorem for non-uniform bottom-up decomposition schemes and the *unbalanced* version of the LPMD problem.

Acknowledgements

The authors thank Unni Narayanan for help with obtaining the software package used in [6], for helpful discussion and the anonymous reviewers for their valuable comments which help improve the presentation of this paper.

References

- [1] J.L. Hennessy and D.A. Patterson, “Computer Architecture : A Quantitative Approach”, 3rd ed., Morgan Kaufmann Publishers, 2003.
- [2] K.S. Khouri, G. Lakshminarayana and N.K. Jha, “IMPACT: A high-level synthesis system for low power control-flow intensive circuits”, in *Proc. of Design, Automation and Test in Euro. 1998*, pp. 848–854, IEEE Press, 1998.
- [3] K. Kim, T. Ahn, S.-Y. Han, C.-S. Kim and K.-H. Kim, “Low-power multiplexer decomposition by suppressing propagation of signal transitions”, in *Proc. of IEEE Int. Symp. on Circuits and Sys. 2001, Vol. 5*, pp. 85–88, IEEE Press, 2001.
- [4] K.-W. Kim, T.W. Kim, T.T. Hwang, S.-M. Kang and C.L. Liu, “Logic transformation for low power synthesis”, *TODAES*, 7(2), pp. 265–283, ACM Press, 2002.
- [5] R. Murgai, R.K. Brayton and A. Sangiovanni-Vincentelli. “Decomposition of logic function for minimum transition activity”, in *Proc. of Euro. Design and Test Conf. 1995*, pp. 404–410, IEEE Press, 1995.
- [6] U. Narayanan, H.W. Leong, K.-S. Chung, and C.L. Liu, “Low power multiplexer decomposition”, in *Proc. of Int. Symp. on Low Power Electronics and Design 1997*, pp. 269–274, IEEE Press, 1997.
- [7] P. Patra and U. Narayanan, “Automated phase assignment for synthesis of low power domino circuits”, in *Proc. of Design Automation Conf. 1999*, pp. 379–384, IEEE Press, 1999.
- [8] R. Panda and F. Najm, “Technology decomposition for low-power synthesis”, in *Proc. of IEEE Custom Integrated Circuits Conf. 1995*, pp. 627–630, IEEE Press, 1995.
- [9] A. Raghunathan, S. Dey and N.K. Jha, “Glitch analysis and reduction in register transfer level power optimization”, in *Proc. of Design Automation Conf. 1996*, pp. 331–336, IEEE Press, 1996.
- [10] S. Thakur, D.F. Wong and S. Krishnamoorthy, “Delay minimal decomposition of multiplexers in technology mapping”, in *Proc. of Design Automation Conf. 1996*, pp. 254–257, IEEE Press, 1996.

- [11] C.-Y. Tsui, M. Pedram and A.M. Despain, “Technology decomposition and mapping targeting low power dissipation”, in *Proc. of Design Automation Conf. 1993*, pp. 68–73, ACM Press, 1993.
- [12] S. Yang and H.W. Leong, “Improved algorithms for low power multiplexor decomposition”, Technical report, School of Computing, National University of Singapore, Singapore, 2006.