Energy Complexity of Logic Level Structures [1]

TR90-010

February, 1990

Akhilesh Tyagi

The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175

Energy Complexity of Logic Level Structures[1]

AKHILESH TYAGI[2]
Department of Computer Science
University of North Carolina
Chapel Hill, N.C.

ABSTRACT

The objective of this paper is to develop a set of combinatorial techniques to compare the energy complexities of the static and dynamic design styles for a set of logic level structures such as a PLA, a gate matrix array and a Weinberger array. We demonstrate these techniques by showing that the average energy consumption of a dynamic PLA exceeds that of a static PLA. We also show that, on average, a dynamic PLA is faster than a static PLA. This is consistent with intuition since one can trade power for delay. In order to prove these results, we deal with a very general class of PLAs. If we restrict this class, we can do more. In particular, we show that the choice of a design style for a data path control PLA depends on the degree of parallelism of the data path. We extend these techniques to the general logic-level structures as gate matrix arrays and Weinberger arrays. A lower bound on the average switching energy of the general logic level structures is also derived. Our techniques are general enough to be applicable to most of the application domains. We believe that our results give a mathematical justification within the limitations of our model for picking one design style over the other for a given application domain.

1 Overview

Several functions have been analyzed for their energy consumption in the VLSI model of computation [1], [9], [10], [11], [16] and [18]. In this paper, we look at a more structured medium of computation, *i.e.*, logic level structures. It is the regularity of a structure that gives us a better handle on the energy complexity analysis. As for the scope of the structural analysis, indeed, all

[1]A preliminary version of this paper appears in the Proceedings of the Stanford Conference on Advanced Research in VLSI, MIT Press, 1987

[2]Research supported by NSF grant number MIP-8806169.

the functions are implemented with a handful of the structured components like PLA, RAM, ROM, multiplexers etc. Our design methodologies teach us to decompose a given function into a few lower level computation and communication primitives that are better understood. For each such primitive, we have a corresponding structure. Thus, this analysis applies to a structural implementation of any function. What are the benefits of structural analysis vis a vis functional analysis? The domain of a structural analysis technique is restricted to a set of well understood structures. Thus the techniques are more powerful, giving rise to a stronger theory.

Our interest in the structure level analysis arose out of a practical need. During the design of the Quarter Horse [8], a 32 bit CMOS microprocessor, we had agreed on the choice of a PLA to implement our control path due to various reasons; flexibility of reconfiguring the control path being the most important one. We were hoping to attain a clock period as close to 50 ns as possible. Since the granularity of timing for an architecture is dictated by the speed of its control path, our first target for the speed improvement was the PLA implementing it. We discarded the option of decomposing the PLA personality matrix into a few smaller and equivalent PLAs as an alternative due to its complexity and nonuniformity. The nonuniformity directly affects our ability to route automatically the control lines from the PLA to the data path. Besides only a PLA with loosely coupled states can benefit from such a decomposition. Thus the obvious choice we were left with was to pick the fastest PLA design style. Due to various factors, our control of the fabrication process being one of them, we did not consider complicated design styles like Cascode Voltage Switch Level (CVSL) [7]. Thus our choice was primarily between a static style or dynamic style PLA implementation. The current state of knowledge did not resolve this question for us. This paper is an attempt to answer the question.

Due to its regularity, a PLA is probably the most popular structure for the implementation of random logic. Its simplicity also makes it the ideal medium for automating the layout process for a random logic expression. Recently, a PLA has been the structure of choice for the control path design in several microprocessors, as in Quarter Horse [8], Mosaic [13], and PP4 [15], primarily due to the ease of reconfiguration. Thus, a significant fraction of silicon area laid out today is devoted to PLA design. It is no surprise, then, that a lot of effort has been devoted to finding ways of optimizing a PLA at the description level (*e.g.* logic minimization [3]) as well as at layout level (*e.g.* folding techniques [4], [6]). However, to the best of our knowledge, there have been no attempts at mathematically analyzing the energy consumption or the delay of this structure. The high performance architectures are reaching a point where power dissipation will be a serious problem. Given this scenario, it is especially fruitful to pay attention to the switching complexity of a PLA, since it is the control path in an architecture which does most of the switching.

In Section 2, we first outline the assumptions we have made about a PLA personality matrix. This is followed by the energy analysis of static and dynamic PLA design styles in Section 3. Section 4 deals with the energy complexity of general logic level structures such as a Weinberger array [19] and a gate matrix array [12]. Section 5 limits the personality matrices of PLAs to a finite state machine and hence estimates the switching more accurately for this class of PLAs. In the last section, we compare the speeds of two design styles.

2 Model

A PLA is completely specified by a collection of sum of products logic expressions [14]. Let there be n input variables denoted by $x_1, x_2, \ldots, x_n$. A *literal* refers to either a variable x_i or the complement of a variable $\overline{x_i}$. A product term consists of a Boolean *and* ($\wedge$) of several literals. A complete product term contains n literals, since each variable occurs in either the complemented form or in its normal form. When a product term contains only $(n - r) < n$ literals, it is said to have r *don't cares.* A PLA output y_i is specified by a Boolean *or* ($\vee$) of several product terms. A PLA personality matrix specifies the sum of products form for all the outputs $y_1, y_2, \ldots, y_l$ in a matrix form. Each product term is also referred to as a *minterm*[3]. Note that this requires that we allow the circuits under consideration to have unlimited fanin and fanout.

We will be counting only the switching of the wires. In asymptotics, the wires dominate the complexity of even a regular structure like a PLA. In a static design, it may seem that when a pull-up is fighting against one or more pull-downs, a considerable amount of energy may be consumed, thus violating this assumption. But, in a static design the pull-up transistors are so weak that the energy consumed by the contention of a pull-up and a pull-down device is a very small fraction[4] of

[3]Although the correct term is *implicant*, we will continue to use *minterm* due to its wide acceptability.

[4]It can be as high as *twice* the energy stored in a wire. However assigning double the cost of a $0 \longrightarrow 1$ transition to a $1 \longrightarrow 0$ transition does not change the relative standings of two styles. To retain simplicity and generality in the model, we assign the same energy cost to both the transitions.

the switching energy of a long minterm wire in poly layer. Thus we will assume that the switching energy of a wire is proportional to its length in the layout.

A PLA satisfies the Uniswitch Model (USM), as defined in Kissin's paper and thesis [10], [11]. In this model, any wire in an acyclic circuit can switch at most once, for an unpipelined computation. To verify this, notice that a PLA does not switch any wire more than once in response to a clock transition, since the state feedback paths (to build a finite state machine) are clocked. Notice, though, that in the dynamic style a wire is first precharged and then is evaluated to logic level zero, hence switching twice. We will still classify this behavior to be in Uniswitch model.

We will be referring to the *state* of a PLA very often in the following discussion. The state of a PLA is given by the values of its minterms and the outputs. Note that the state of a PLA is completely specified by the value of its input bits. Over a complete cycle of operation, the input defines the final values of the minterms and the outputs.

Our objective is to compare the switching energy consumed by the static and the dynamic design styles. Instead, we compare only the total amount of switching between two design styles. Note that once a PLA personality matrix is fixed, the lengths of both the minterm and output wires are fixed. And the switching energy just equals the product of the amount of switching and the wire length. Thus the wire lengths can be factored out for the purpose of comparison since they are the same for both the design styles. This also separates the inherent switching complexity issues from the layout issues. The switching amount is a property of the personality matrix, but the wire lengths depend on the particular layout. The clever layout techniques like folding – to reduce the wire lengths – could very well be applied across both the design styles. This validates our strategy of comparing only the switching.

A PLA is used in a wide spectrum of applications. This makes it very hard to say anything about the distribution of state transition probabilities. In order to make the problem tractable, we restrict a PLA personality in the following way. We assume that the domain of PLAs we are considering has the property that, given s_i is the current state, the next state could be any of the 2^n states with equal probability. Of course, in a real PLA certain state transitions are more heavily favored than others. We do not have sufficient information about a PLA personality to assign nonuniform weights to the state transitions. A more complicated modeling could attempt to look at it as a Markov process. However, this is an attempt to capture every kind of PLA into a single unified model. Alternatively, one could build a different model of a PLA for each domain of applications. We take the later approach in Section 5, by modeling a control PLA for a data path.

In what follows, m is the number of minterms, n is the number of inputs, and, l is the number of outputs for the PLA under consideration. Thus the total area of a PLA under either implementation, is $\Theta(m\,(n+l))$. Having set the ground rules, we are ready to analyze the energy complexity of PLA design styles.

3 PLA Energy Complexity

Note that the input switching is uniform to both the design styles. However the way the minterms and the outputs are evaluated is very different. Thus we need to compare only the minterm and output switching. We will consider both NAND and NOR style dynamic and static PLAs. A $p-$type precharge scheme is more commonly used in today's CMOS dynamic design styles. For this reason, in our discussion a dynamic style PLA refers to a $p-$type precharge scheme dynamic PLA, where a node is precharged to logic level 1.

We develop some notation before we go any further. Let M be the set of minterms. More specifically, M has integer elements in the range $[0, m-1]$. Notice that M is associated with the personality matrix A for a PLA for a function f. From now on, the personality matrix A and the function f will be implicit in the notation and will not be explicitly stated.

Definition 1 *An onset for a state s_i, denoted by $On(s_i) \subseteq M$ is the set of minterms that are active (on) in the state s_i for a PLA computing a function f.*

We will think of $On(s_i)$ as a bit vector of length m. Let $\sim On(s_i)$ denote the bitwise complement of $On(s_i)$. $\vec{a} \cap (\cup) \vec{b}$ denotes the bitwise '$\wedge(\vee)$' of $\vec{a}$ and $\vec{b}$. The *weight* of a bit vector $\vec{a}$, $|\vec{a}|$, is the number of bit positions with entry 1 in $\vec{a}$. What is the domain of values that $On(s_i)$ can assume? To answer this question let us define $ON(\vec{x})$ to be the set of active minterms with the input vector being $\vec{x}$. Let $\mathcal{M} = \cup_{\vec{x} \in \{0,1\}^n} ON(\vec{x})$ be the union of active minterm sets over all the input vectors. Clearly the domain of values for $On(s_i)$ is the set $\mathcal{M}$.

Similarly, let O be the set of output bits with integer elements in the range $[0, l-1]$. Let $Out(s_i)$ denote the set of output bits that are active (on) in state s_i. At times, we will think of $Out(s_i)$ as a $l-$bit vector. Let $\mathcal{O} = \cup_{s_i} Out(s_i)$ be the domain of values for $Out(s_i)$.

We count only the average switching, which is the total switching averaged over all the 2^{2n} state pairs. Note that this allows a PLA to stay in the same state in the next cycle. To determine the average switching, all that we can assume is that the input values are uniformly distributed from the set $\{0,1\}^n$. What can we say about the vectors in $\mathcal{M}$ and $\mathcal{O}$? A PLA specifies a mapping from the set of input vectors $\{0,1\}^n$ to the sets $\mathcal{M}$ and $\mathcal{O}$. The elements of $\mathcal{M}$ have certain characteristics determined by the fact that an *and plane* implements an *and* of the input bits. This mapping forms the basis of the minterm switching analysis for the static and dynamic design styles in the following subsection. Subsection 3.2 deals with the switching of the output lines. Section 4 contains the analysis of average switching energy of two design styles if we did not know anything about the mapping between the input values and the set $\mathcal{M}$.

3.1 And Plane

In the following, we show that the average switching in the *and plane* of a NOR style dynamic PLA is given by $\frac{(2^{n-r}-1)}{2^{n-r-1}}m$, where every minterm on average contains r *don't cares*. A NAND style dynamic design, on the other hand, requires $\frac{m}{2^{n-r-1}}$ switching. For a static PLA, both NAND and NOR styles lead to an average switching of $\frac{(2^{n-r}-1)}{2^{2(n-r)-1}}m$. Notice that on average the NOR style dynamic PLA consumes 2^{n-r} times as much energy as a NOR or NAND style static PLA. In contrast, a NAND style dynamic PLA asymptotically consumes the same order of energy as a static style PLA.

In the proofs that follow, we need to determine the average number of 1's in a minterm set. The following discussion addresses this issue. Let the input values be chosen from $\{0,1\}^n$ with a uniform distribution. Since each minterm, on average, contains r *don't cares*, a minterm must be shared by 2^r input values, *i.e.* for a given minterm 2^r distinct input values will turn it *on*. One implication of this fact is that each minterm set in a NOR (NAND) style PLA, on average, contains $m/2^{n-r}$ $\left(\left[m - \frac{m}{2^{n-r}}\right]\right)$ 1's. The following lemma proves it.

Lemma 1 *Let each minterm, on average, contain r don't cares. Then the average number of 1's in a minterm set $\vec{x} \in \mathcal{M}$, $|\vec{x}|$, is given by $m/2^{n-r}$ for a NOR style* and plane. *In a NAND design, the average number of 1's is given by $m - \frac{m}{2^{n-r}}$.*

Proof: Consider a table of the following form. Each row corresponds with a minterm and each column corresponds with an input value from $\{0,1\}^n$. Thus there are m rows and 2^n columns.

The position (i, j) in this table contains a 1 if the minterm in the ith row evaluates to 1 for the input assignment in column j. In the NOR case, only one assignment of values to the literals in a minterm evaluates it to 1, *i.e.*, when all of $n - r$ literals in a minterm are set to zero. Since each minterm, on average, has $n - r$ literals, it evaluates to 1 for 2^r input assignments. Hence each row on average contains 2^r 1's. The total number of 1's in the table then is $m2^r$. This averaged over 2^n input values provides the average number of 1's in a minterm set (in a column in this table) to be $m/2^{n-r}$ for the NOR case.

In the NAND case, a minterm evaluates to 0 only when each of its $(n - r)$ literals is assigned a 1. Thus 2^r out of 2^n input assignments give rise to a 0 in a row. Thus the total number of 1's in a row is $2^n - 2^r$ with the total number of 1's in the table being $(2^n - 2^r)m$. Averaging it over 2^n input values gives the average number of 1's in a NAND style *and plane* minterm as $m - \frac{m}{2^{n-r}}$. This can also be rewritten as $m \frac{(2^{n-r}-1)}{2^{n-r}}$.

□

We will be using this lemma many times in the following proofs. We start the analysis with the dynamic case, which is simpler.

3.1.1 Dynamic Design Style

The total switching from state s_p to s_c is denoted by $Sw(s_p, s_c)$. Let us estimate the switching, $Sw(s_p, s_c)$, in the precharged logic. In a dynamic logic, there is no coupling between two states s_p and s_c. State s_p has no influence on the switching in State s_c and State s_c does not affect the switching in State s_p. The two independent components of switching due to s_c and s_p are:

1. Precharging the minterms that were discharged in the previous state. This number is $m - |On(s_p)|$.

2. Discharging the minterms that are not active in the current state. This number is $m - |On(s_c)|$.

Thus, the expression for total switching in the dynamic logic case is given by:

$$Sw(s_p, s_c) = 2m - |On(s_p)| - |On(s_c)| \tag{1}$$

Notice that the worst case switching in this case could be almost as large as $2m$. The following lemma determines the average switching in the dynamic case.

Lemma 2 *The average switching for a NOR (NAND) style dynamic PLA* and plane *with r don't cares per minterm is* $\frac{(2^{n-r}-1)}{2^{n-r-1}}m \ \left(\frac{m}{2^{n-r-1}}\right)$.

PROOF: The average switching is given by $2m \ - \ 2\,|\bar{x}|$ by Equation 1, where $|\bar{x}|$ denotes the average weight of an element in $\mathcal{M}$. For the NOR case, the average weight of an element in $\mathcal{M}$ is given by $m/2^{n-r}$ by Lemma 1. Hence the NOR case average switching is $2m\left(1 \ - \ \frac{1}{2^{n-r}}\right)$. Rearranging the terms, we get $\frac{(2^{n-r}-1)}{2^{n-r-1}}m$. In the NAND case, from Lemma 1 the average weight is $m\,(2^{n-r}-1)\,/\,2^{n-r}$. Hence the average switching is $\frac{m}{2^{n-r-1}}$.

$\square$

Note that $n-r$ can take values in the range $[1, n]$. At least one literal is required to specify a nontrivial minterm, giving rise to the lower end of the range. In the other extreme, all n literals of a minterm may be specified. The dynamic NOR style average switching then ranges from m to $\frac{(2^n-1)}{2^{n-1}}m$. The upper range is approximately $2m$. The NAND style average switching ranges from $\frac{m}{2^{n-1}}$ to m. Notice that this time the upper range of $n-r$ gives rise to the lower range of average switching. Also notice that this gap in the average switching of NAND and NOR dynamic styles is not an intrinsic one. Our assumption regarding $p-$type precharge leads to this gap. In an $n-$type precharge dynamic logic this relationship between NOR and NAND average switching will be inverted. The NOR (NAND) style then will have $m/2^{n-r-1}$ $(m\,(2^{n-r}-1)/2^{n-r-1})$ average switching. Now let us look at the static design style PLAs.

3.1.2 Static Design Style

Let us consider the switching for a transition from a previous state, s_p, to a current state, s_c. Arguing in a similar way as for Equation 1, one can see that the total switching, $Sw(s_p, s_c)$, is given by the following equation.

$$Sw(s_p, s_c) \ = \ |(On(s_p) \cap (\sim On(s_c)))| \ + \ |((\sim On(s_p)) \cap On(s_c))| \qquad (2)$$

From this equation, the worst case switching can be at most m when two minterm sets differ in all the bit positions. Contrast this with the dynamic case where the worst case switching could have been almost as high as $2m$. Note that for each pair of minterm sets $\vec{u}, \vec{v} \in \mathcal{M}$, we need to calculate $|\vec{u} \cap \sim \vec{v}| + |\sim \vec{u} \cap \vec{v}|$. Also notice that this quantity equals exactly the number of bit positions that $\vec{u}$ and $\vec{v}$ differ in. This is the Hamming distance of $\vec{u}$ and $\vec{v}$, which we shall denote by $h(\vec{u}, \vec{v})$.

Let us establish some notation before we go any further. The total Hamming distance of a set S, denoted by $h_t(S)$, is $\sum_{\vec{x},\vec{y} \in S} h(\vec{x}, \vec{y})$. The average Hamming distance of a set S, denoted by $h_a(S)$, is $h_t(S) \,/\, |S|^2$. Our basic strategy will be to count the total Hamming distance of the set $\mathcal{M}$, $h_t(\mathcal{M})$ first and then to divide it by the total number of state pairs to derive $h_a(\mathcal{M})$. The following lemma does that.

Lemma 3 *The average switching for both NOR and NAND style static PLA* and plane *with r don't cares per minterm is given by* $\frac{(2^{n-r}-1)}{2^{2(n-r)-1}}m$.

PROOF: Note that 2^n input values are mapped into 2^{n-r} minterm sets. Hence the cardinality of the set $\mathcal{M}$ must be 2^{n-r}.

We wish to determine the average Hamming distance of $\mathcal{M}$. Consider a row-wise enumeration of all the m-bit vectors in $\mathcal{M}$. Thus position (i, j) lists the jth bit of the ith element of $\mathcal{M}$. There are 2^{n-r} rows and m columns in this enumeration. By Lemma 1, for the NOR case a row in this enumeration on average has $m/2^{n-r}$ 1's. Then the total number of 1's in this table is m. Hence the average number of 1's per column is 1. Thus each column, on average, has one 1 and $(2^{n-r} - 1)$ 0's. Let us count each column's contribution towards the total Hamming distance. The single 1 differs from $2^{n-r} - 1$ rows containing 0 in that column contributing $(2^{n-r} - 1)$ towards the total Hamming distance. Similarly, each of $2^{n-r} - 1$ 0's contributes a 1 towards the total Hamming distance. Thus each column has a Hamming distance of $2\,(2^{n-r} - 1)$ giving a total Hamming distance of $2m\,(2^{n-r} - 1)$. This quantity averaged over $2^{2(n-r)}$ row pairs provides an average Hamming distance of $\frac{(2^{n-r}-1)}{2^{2(n-r)-1}}m$. Recall that the average switching is given by the average Hamming distance. The NAND case has the same average Hamming distance.

□

Once again, letting $(n-r)$ range over $[1, n]$ gives $\left[\frac{2^n-1}{2^{2n-1}}m,\; m/2\right]$ for the average switching range. The lower bound goes to $m/2^{n-1}$ in the limit. Notice that the expression for the average dynamic switching in Lemma 2 is about 2^{n-r} times as large as the one for a static PLA. The reason for this gap is the independence of two states in the dynamic case. In other words, a dynamic design has no memory. The current state s_c does not remember, and hence does not benefit from any information about the previous state, s_p. Now let us consider the switching in an *or plane*.

3.2 Or Plane

This subsection determines the average switching in an *or plane*. Recall that there are l output lines and m minterm lines in the *or plane*. Let each output function contain s minterms on average. This implies that the average number of 1's in each column on the output side of the PLA personality matrix is s. Note that $1 \leq s \leq m$. Then we will prove that the average switching in the *or plane* of a NOR-NOR dynamic PLA is $2l\left[1 - \left(1 - 2^{-(n-r)}\right)^s\right]$. A NAND-NAND dynamic PLA *or plane* switches $2l\left[\left(1 - 2^{-(n-r)}\right)^s\right]$ times. For a static style *or plane*, both NAND and NOR designs lead to $2l\left(1 - 2^{-(n-r)}\right)^s\left[1 - \left(1 - 2^{-(n-r)}\right)^s\right]$ switching. We will refer to the collection of l output functions by the term *output word*. We will first prove a lemma analogous to Lemma 1 estimating the average number of 1's in an output word.

Lemma 4 *Let the average number of don't cares per minterm be r and the average number of minterms per output function be s. Then the average number of 1's in an output word $\bar{y} \in \mathcal{O}$, $|\bar{y}|$, is given by $\left(1 - 2^{-(n-r)}\right)^s l$ for a NOR-NOR PLA. In a NAND-NAND design, the average number of 1's is given by $\left[1 - \left(1 - 2^{-(n-r)}\right)^s\right] l$.*

PROOF: The proof is very similar to the proof of Lemma 1. Let us concentrate on a particular output bit y_j. Since every output function contains s minterms on average, we can draw a table as in Lemma 1 with s minterms of y_j along the rows and 2^n input assignments along the columns. Let us first deal with the NOR-NOR case. Each of these s rows will contain exactly 2^r 1's and $2^n - 2^r$ 0's. Hence the probability of a given minterm (a row) being a 0 is $1 - 2^{-(n-r)}$ and the probability of its being a 1 is $2^{-(n-r)}$. For a NOR style *or plane*, an output bit evaluates to 1 if and only if all of its s minterms evaluate to 0. The probability of all the s minterms being 0 is $(1 - 2^{-(n-r)})^s$. Thus the expected number of 1's in an output word is $(1 - 2^{-(n-r)})^s l$.

Let us now consider the NAND-NAND case. Each minterm evaluates to 0 for 2^r input values and to 1 for $2^n - 2^r$ input values. An output bit is 0 if and only if all its minterms evaluate to

1. The probability of this event is $(1-2^{-(n-r)})^s$. Hence the expected number of 1's in an output word is $[1 - (1-2^{-(n-r)})^s]\, l$.

□

Now let us analyze both dynamic and static style *or planes* using the same techniques as in Subsection 3.1.

3.2.1 Dynamic Design Style

Once again, the switching in a transition from a state s_p to a state s_c is given by an equation analogous to Equation 1.

$$Sw(s_p, s_c) = 2l - |Out(s_p)| - |Out(s_c)| \tag{3}$$

This equation along with Lemma 4 gives rise to the following lemma.

Lemma 5 *Let a PLA have r don't cares per minterm and s minterms per output on average. Then the average switching for a NOR-NOR (NAND-NAND) style dynamic PLA* or plane *is* $2l\left[1 - \left(1-2^{-(n-r)}\right)^s\right]$ $\left(2l\left[\left(1-2^{-(n-r)}\right)^s\right]\right)$.

3.2.2 Static Design Style

The switching between two states s_p and s_c is given by an equation similar to Equation 2.

$$Sw(s_p, s_c) = |(Out(s_p) \cap (\sim Out(s_c)))| + |((\sim Out(s_p)) \cap Out(s_c))| \tag{4}$$

Thus we need to estimate the average Hamming distance of the set $\mathcal{O}$. The following lemma does that.

Lemma 6 *The average switching for both NOR and NAND style static PLA* or plane *with r don't cares per minterm and s minterms per output is given by* $2l\left(1-2^{-(n-r)}\right)^s\left[1 - \left(1-2^{-(n-r)}\right)^s\right]$.

PROOF: Let there be R elements in $\mathcal{O}$. We have to estimate the average Hamming distance of the set $\mathcal{O}$, $h_a(\mathcal{O})$. Once again, let us enumerate the elements of $\mathcal{O}$ in R rows. Each column corresponds with one output bit. By Lemma 4, each row has $l\left(1-2^{-(n-r)}\right)^s$ 1's for the NOR-NOR case. Hence the average number of 1's per column is $R\left(1-2^{-(n-r)}\right)^s$ and the average number of 0's per column is $R[1 - \left(1-2^{-(n-r)}\right)^s]$. Thus the total Hamming distance of the set $\mathcal{O}$ is $2lR^2\left(1-2^{-(n-r)}\right)^s[1 - \left(1-2^{-(n-r)}\right)^s]$. Averaged over R^2 combinations, this gives an average Hamming distance of $2l\left(1-2^{-(n-r)}\right)^s\left[1 - \left(1-2^{-(n-r)}\right)^s\right]$. The NAND-NAND case also leads to the same result.

□

Recall that $(n-r)$ takes on values in the range $[1,n]$ and s is in the range $[1,m]$. Notice once again that the static style consumes less energy than either NOR-NOR or NAND-NAND dynamic style for the *or plane* as well. On the other hand, NAND-NAND dynamic style *or plane* seems to consume more energy than a NOR-NOR dynamic style *or plane*.

We can add up the average switching of both the *and plane* and *or plane* to get the following corollaries.

Corollary 1 *For a dynamic PLA in NOR-NOR (NAND-NAND) style, the average switching is* $\frac{(2^{n-r}-1)}{2^{n-r-1}}m + 2l\left[1 - \left(1-2^{-(n-r)}\right)^s\right]\ \left(m/2^{n-r-1} + 2l\left[\left(1-2^{-(n-r)}\right)^s\right]\right)$.

Corollary 2 *For a static PLA, the average switching is given by* $\frac{(2^{n-r}-1)}{2^{2(n-r)-1}}m + 2l\left(1-2^{-(n-r)}\right)^s$ $\left[1 - \left(1-2^{-(n-r)}\right)^s\right]$.

Notice that the static design consumes at least half as much energy as either of NOR-NOR or NAND-NAND dynamic PLAs for any values of r and s. A more accurate comparison can be obtained by assuming the lengths of minterms to be proportional to $2n+l$ and the lengths of the output wires to be proportional to m. Then we get the following corollaries.

Corollary 3 *For a dynamic PLA in NOR-NOR (NAND-NAND) style, the average switching energy is proportional to* $\frac{(2^{n-r}-1)}{2^{n-r-1}}m(2n+l) + 2ml\left[1 - \left(1-2^{-(n-r)}\right)^s\right]$ $\left(m(2n+l)/(2^{n-r-1}) + 2ml\left[\left(1-2^{-(n-r)}\right)^s\right]\right)$.

Corollary 4 *For a static PLA, the average switching energy is given by* $\frac{(2^{n-r}-1)}{2^{2(n-r)-1}} m(2n+l) + 2ml\left(1-2^{-(n-r)}\right)^{s}\left[1 - \left(1-2^{-(n-r)}\right)^{s}\right]$.

This shows that a static style PLA has a lower average energy consumption than a dynamic style PLA. Within the dynamic style, the choice between a NOR-NOR or NAND-NAND design depends on two parameters: the average number of *don't cares* per minterm r and the average number of minterms per output s.

4 General Logic Level Structure Energy Complexity

In the preceding analysis, we had fixed the mapping between the set of input values $\{0,1\}^n$ and the sets $\mathcal{M}$ and $\mathcal{O}$. This in turn fixed one characteristic of the sets $\mathcal{M}$ and $\mathcal{O}$: the average weight of their elements. If we do not bind this mapping, we can get a comparison of two design styles for a general logic level structure.

We investigate the following question. Consider a layout structure implementing a multilevel Boolean function with n input bits and m output bits. (Note that a PLA implements a 2-level, AND-OR, Boolean function.) The two structures that are commonly used for this purpose are Weinberger arrays [19] and gate matrix arrays [12]. Notice that these structures do not fix the mapping function between the input values and the output values to be *nand* or *nor*. We wish to derive a value for average switching of output bits in these structures. Note that we are not counting the switching of intermediate values. (An intermediate value is a function of input values that is not an output bit. For example, in $f = (x_1x_3 + x_2)(x_1x_4)$, x_1x_3 is an internally generated *intermediate value*.) However the same results about relative energy complexity of dynamic and static styles also apply to the intermediate values. Thus couting the switching of the intermediate values will only magnify the relative standings of the two design styles.

Can the switching count still be used as a basis for switching energy comparison between two design styles? Admittedly not all the output lines in a Weinberger array or a gate matrix array have the same length. But for a given layout, these lengths are fixed. Thus the average switching still forms a basis for switching energy comparison between static and dynamic design styles. In most of the following discussion we also assume that no output bits are shared, *i.e.*, each input value gets mapped to a unique output value.

The following subsection estimates the average switching of the output bits when $m = kn$ for a small constant $k > 0$. What happens if m is allowed to be arbitrarily large with respect to n? In Subsection 4.2 we derive some lower bounds on the average switching for a general logic level structure for this case.

4.1 General Logic Level Structure Switching

We show the following in this subsection. On average, m output bits switch in a dynamic style general logic level structure, while only $m/2$ bits switch for the static case. In the following discussion, n is the number of input bits and m is the number of output bits. The input values are uniformly distributed from the set $\{0,1\}^n$. The state of a logic level structure is, once again, well defined and is completely determined by the input value[5]. Let O be the set of output bits with elements from $[0, m-1]$. $Out(s_i)$ is the set of output bits that are *on* in state s_i. The domain of values of $Out(s_i)$ is denoted by $\mathcal{O}$. Let us start with the dynamic case.

4.1.1 Dynamic Design Style

The expression for total switching in the dynamic logic case is very similar to Equation 1. The switching between two states s_p and s_c is given by:

$$Sw(s_p, s_c) = 2m - |Out(s_p)| - |Out(s_c)| \qquad (5)$$

We start with a simple case, when $m \leq n$. Since $2^m \leq 2^n$, this fixes the set $\mathcal{O}$ to be $\{0,1\}^m$ and does not leave any room for the adversary to pick the vectors of $\mathcal{O}$ cleverly. Note that although we had assumed that the output words are not shared, in this case the sharing will be necessary. For this reason, we also assume that 2^m output words are uniformly distributed. We will not need this assumption for other lemmas and theorems.

Lemma 7 *The average switching for a dynamic logic structure with $m \leq n$ is m.*

Proof: First, we have to sum $|Out(s_p)| + |Out(s_c)|$ over all pairs (s_p, s_c). Let us visualize a truth table (a lexicographically ordered enumeration) of m-bit vectors, as we state the proof. For a fixed

[5]No feedback paths are permitted in a multilevel logic description.

$Out(s_p)$ we fix a row and count each of the 2^m rows with it. For 2^m possible values of $Out(s_p)$, each row is counted 2^m times as $Out(s_c)$. This sum taken over all the rows in the truth table equals $(2^m) \times$ *(total number of 1's in the truth table).* Note that the total number of 1's in the table is $m\,2^{m-1}$. When a row corresponds with $Out(s_p)$, it is counted once in conjunction with each of the 2^m possible $Out(s_c)$. It provides another term of $2^m\,m\,2^{m-1}$. Thus the total switching is $m\,2^{2m}$ and the average switching is m. □

Most of the functions implemented with a general logic level structure in practice will have a very few output bits and will be covered by the previous lemma. But very often, the number of output bits m is a small constant multiple of the number of input bits n. As the next generalization, we allow for m to be that large. Hence, let us consider the case when $m = kn$ for $k > 1$. As we will see in the proof of the following theorem, it makes a difference whether $\mathcal{O}$ is closed under complement, *i.e.*, if $\vec{u} \in \mathcal{O}$ then $\sim \vec{u} \in \mathcal{O}$. We will call $\vec{u} \in \mathcal{O}$ an unpaired output word if $\sim \vec{u} \notin \mathcal{O}$.

Theorem 1 *Let the number of outputs $m = kn$ with $k > 1$. Let there be $2p$ unpaired output words in the set of assigned output words $\mathcal{O}$. Then the average switching between any two states, $Sw(s_i, s_j)$ is m.*

PROOF: This time $\mathcal{O}$ contains 2^n output words. Recall from the proof of Lemma 7 that each row is counted 2^n times as $Out(s_p)$ and 2^n times as $Out(s_c)$. Each pair of complemented output words contributes m 1's to the truth table. The number of complemented pairs is given by $2^{n-1} - p$. Thus the number of 1's in the truth table due to complemented pairs is $m\,(2^{n-1} - p)$. To count the number of 1's in the unpaired $2p$ output words, we estimate the average number of 1's from a collection of sets containing $2p$ unpaired $m-$bit vectors. These sets are either of the form $0\{0,1\}^{m-\log(2p)-1}\{0,1\}^{\log(2p)}$ or of the form $1\{0,1\}^{m-\log(2p)-1}\{0,1\}^{\log(2p)}$. This expression specifies $2^{m-\log(2p)}$ sets of $2p$ vectors each. Notice that since the MSB is fixed to a 0 or a 1, all the $2p$ vectors in such a set are unpaired. The average number of 1's in $2p$ least significant bit positions is $\log(2p)/2$ by the counting technique of Lemma 7. Similarly, the middle $m - \log(2p) - 1$ bit positions contribute $[m - \log(2p) - 1]/2$ to the average number of 1's. The most significant bit position can have a 1 with probability $1/2$. This says that the average number of 1's per vector in such a set is $m/2$. Hence, the total number of 1's in the truth table is $m\,(2^{n-1} - p) + m\,p$, which equals $m\,2^{n-1}$. Along the lines of the proof of Lemma 7, the total switching is $m\,2^{2n}$ and the average switching is m. □

Having analyzed the dynamic style, let us look at the static case.

4.1.2 Static Design Style

The expression for switching count in an output word is analogous to Equation 2. The switching from state s_p to s_c, $Sw(s_p, s_c)$, is given by:

$$Sw(s_p, s_c) \; = \; |(Out(s_p) \cap (\sim Out(s_c)))| \; + \; |((\sim Out(s_p)) \cap Out(s_c))| \tag{6}$$

This is exactly the average Hamming distance of the set $\mathcal{O}$. We will show that the maximum average switching attainable is $m/2$ for a reasonable logic structure. A reasonable logic structure is a function with m being equal to $k\,n$, where $k > 0$ is a very small constant; since in practice that is how m and n relate.

The expression in Equation 6 is symmetric with respect to the pair (s_p, s_c). That tempts us to count the overlaps between all the pairs of subsets of O and double it to get the switching totaled over all pairs (s_i, s_j). When can that be done? The domain of values for $Out(s_i)$ and $\sim Out(s_j)$ is $\mathcal{O}$. If $\mathcal{O}$ were closed under complement then we could collapse the second term in Equation 6 on top of the first term. To make things even simpler we start with the assumption that $m \leq n$. Once again, this case forces the output words to be shared. Hence for this case, we assume that the elements of $\mathcal{O}$ are uniformly distributed. This forces the set $\mathcal{O}$ to be $\{0,1\}^m$. We will relax these assumptions as we go on. In addition to the sum of overlaps between all the unordered pairs of the subsets of O, there is another term due to self overlap.

Once again, our basic strategy will be to count the total Hamming distance of the set $\mathcal{O}$. It is time to prove the simplest result.

Lemma 8 *The average switching for a static logic structure with $m \leq n$ is given by $m/2$.*

Proof: Let us first count the total Hamming distance (also referred to as *overlap*). Consider a lexicographically ordered enumeration of all the 2^m output words. Each row in this enumeration is assigned to a unique set of 2^{n-m} states. Thus the average switching from a previous state s_p to a current state s_c is given by the average Hamming distance of any two rows in this enumeration. Let us fix a row as the current output word (an m-bit vector $\vec{x}$). For any column j (bit position

j), this row (vector $\vec{x}$) differs from 2^{m-1} other rows (vectors). Thus each bit position contributes a Hamming distance of 2^{m-1}. For a fixed initial state, then, the total Hamming distance is given by $m\,2^{m-1}$. Any of 2^m m-bit vectors could be the initial state vector. Hence, the total Hamming distance is $m\,2^{2m-1}$. Dividing it by the number of state pairs 2^{2m} gives the average switching of $m/2$.

□

We illustrate the previous lemma by an example. Consider the following table of output values for $m = n = 2$.

o_0	0	0
o_1	0	1
o_2	1	0
o_3	1	1

Consider the transitions from o_0 to any other state o_0, o_1, o_2 or o_3. o_0 differs from o_2 and o_3 in bit position 1 contributing 2 to the total Hamming distance. Similarly o_0 differs from o_1 and o_3 in bit position 0 contributing another 2 to the Hamming distance. It can be verified that each of o_1, o_2 and o_3 also contribute 4 to the Hamming distance. Thus the total Hamming distance is 16. We divide the total Hamming distance by the number of state pairs giving $16/16 = 1$ as the average switching.

The next generalization comes from considering the case when $k > 1$. For each one of 2^n input states, we need to assign a distinct output word from a set of $2^m > 2^n$ output words (Now we assume that there is no sharing of minterms). Recall that $\mathcal{O}$ is the set of 2^n output words that are assigned to one of the states. There are two cases, one when $\mathcal{O}$ is closed under bitwise complement, and the other, when it is not. Let us consider the first case. There are 2^{n-1} complement pairs in $\mathcal{O}$. Intuitively, it seems likely that given a large number, 2^m, of output words to choose from, we could intelligently pick 2^n output words to lower the switching below Lemma 8 level. But as the following theorem shows, the switching only gets worse. The intuitive reasoning behind this phenomenon is as follows. Each complement pair has m 1's between two of them. There are 2^{n-1} such pairs. Thus the total number of 1's in the elements of $\mathcal{O}$ is $m\,2^{n-1}$. The total number of bits in these entries is $m\,2^n$. There are exactly half 1 entries and half 0 entries. Thus, the expected Hamming distance between any two elements of $\mathcal{O}$ is $m/2$. Recall that $h(\vec{u}, \vec{v})$ denotes the Hamming distance between $\vec{u}$ and $\vec{v}$. We state and prove the theorem next.

Theorem 2 *Let the number of output bits $m = kn$ with $k > 1$. Let the set of assigned output words $\mathcal{O}$ be closed under bitwise complement. Then the average switching between any two states, $Sw(s_i, s_j)$, is given by $m/2$.*

PROOF: This time we will be counting the Hamming distance between complement pairs. Let us consider two complement pairs $(\vec{u}, \sim \vec{u}) \in \mathcal{O}$ and $(\vec{v}, \sim \vec{v}) \in \mathcal{O}$. We claim that the transitions from the output words from a complement pair $(\vec{u}, \sim \vec{u})$ to a complement pair $(\vec{v}, \sim \vec{v})$ have a Hamming distance of $2m$. There are two cases as follows.

$\vec{u} = \vec{v}$ **or** $\vec{u} = \sim \vec{v}$: In this case, notice that the Hamming distance between $\vec{u}$ and $\sim \vec{u}$ is m. Thus the transitions $\vec{u} \longrightarrow \sim \vec{u}$ and $\sim \vec{u} \longrightarrow \vec{u}$ contribute $2m$ to the total Hamming distance.

$\vec{u}, \sim \vec{u} \neq \vec{v}$: First consider the transition $\vec{u} \longrightarrow \vec{v}$. Let us assume that the Hamming distance of $\vec{u}$ and $\vec{v}$ is q. Then the Hamming distance of $\vec{u}$ and $\sim \vec{v}$ is $m - q$. Thus $h(\vec{u}, \vec{v}) + h(\vec{u}, \sim \vec{v})$ is m. Similarly, $h(\sim \vec{u}, \vec{v}) + h(\sim \vec{u}, \sim \vec{v})$ is m. Hence, all the transitions between two complement pairs contribute $2m$ to the total Hamming distance.

There are 2^{n-1} complement pairs. Thus, the total number of distinct complement pairs is given by 2^{2n-2}. Hence the total Hamming distance is $m\,2^{2n-1}$. Thus the average switching is $m/2$.

$\square$

Now we consider the second case when the set $\mathcal{O}$ is not closed under complement. This gives rise to a range for the average switching depending on the number of unpaired output words, $2p$, in $\mathcal{O}$.

Theorem 3 *Let $m = kn$ with $k > 1$. Let there be $2p$ unpaired output words in $\mathcal{O}$. The average switching in this case is*

$$\frac{m\,2^n\,(2^{n-1} - p)}{2^{2n}} + \frac{2p^2 \log(2p)}{2^{2n}}.$$

PROOF: We have to consider the interactions between three groups of the output words belonging to $\mathcal{O}$. Let set A consist of the output words that have their complements in $\mathcal{O}$. The cardinality of A is $2^n - 2p$. Let $2p$ unpaired output words belong to set B. Let the bitwise complements of $2p$ output words in B belong to set C. There are three terms in the expression for the total switching.

- The first one is due to internal overlap between the elements of A. This analysis is similar to the one in the proof of Theorem 2. Thus, the total overlap is $2m\,(2^{n-1} - p)^2$.
- The second term is due to the interaction between set A and the sets B and C, due to the symmetry of Expression 6. Its contribution is $m(2^{n-1} - p)2p$.
- The third term approximates the contribution of the interaction of sets B and C. We calculate this contribution in the following discussion.

This term's contribution to the total switching is $H = \sum_{\vec{u},\vec{v}\in B} h(\vec{u},\vec{v})$, which is the Hamming distance of the set B. We cannot really estimate H, the Hamming distance of a set for an arbitrary set B. However, since we are interested in the average behavior, we can calculate the average Hamming distance of a set picked from a given collection of sets. In this case, all the sets in this collection must not be closed under complement and their cardinality must be $2p$. Only $\log(2p)$ bits are required to specify a collection of $2p$ vectors. Let us fix the other $m - \log(2p)$ bits to zero. Note that for $2p < 2^m$, at least one bit will be fixed to zero thereby guaranteeing that no set in this collection is closed under complement. We can specify this collection of sets as $\{0^{m-\log(2p)}\{0,1\}^{\log(2p)}\}$. Let us estimate the Hamming distance of this collection of sets.

Let one be the least significant bit position and m the most significant bit position. For each bit position $1 \le j \le \log(2p)$, a bit vector $\vec{u}$ differs from $2^{\log(2p)-1} = p$ vectors. But for the bit positions $\log(2p) + 1 \le j \le m$, each vector contains a zero. Thus the left $m - \log(2p)$ columns do not contribute anything to the Hamming distance of this set. The right $\log(2p)$ columns contribute p each towards the Hamming distance. Thus, the Hamming distance of a set of $2p$ distinct m-bit vectors is $\log(2p)\,p\,2p$. It has to be averaged over $4p^2$ pairs. This gives us an average overlap of $\log(2p)/2$ and a total Hamming distance of $2p^2\log(2p)$.

All the three terms combined and averaged over 2^{2n} state pairs give us the average switching in the following form.

$$\frac{m\,2^n\,(2^{n-1} - p)}{2^{2n}} + \frac{2p^2\,\log(2p)}{2^{2n}}$$

□

The first term in Theorem 3, $\frac{m\,2^n\,(2^{n-1}-p)}{2^{2n}}$ dominates as the number of unpaired output words $2p$ tends to zero, and the average switching tends to $m/2$. As the ratio of unpaired output words rises, ($p \longrightarrow 2^{n-1}$), the second term dominates in the average switching expression. With this extreme, in the limit, the average switching is $n/2$. Thus, the average switching is a monotonic nonlinear function of p.

To summarize, a static logic structure has a maximum average switching of $m/2$ (when $\mathcal{O}$ is closed under complement). Even in the general case, the average switching of a static structure is in the range $[n/2,\, m/2]$. In the following subsection, we derive asymptotic lower bounds on the amount of switching in dynamic and static designs for a general logic level structure as a function of m and n.

4.2 Lower Bounds on Average Switching

We estimated the average switching for a PLA as a function of both m and n in Subsection 3. There was no restriction on how large or small m could be with respect to n. (Although $m \geq n - r$ since at least 2^{n-r} states are required). But our discussion of the average switching for a logic level structure so far has been very pragmatic in the following sense. The estimation of average switching for a logic level structure was limited to the practical domain where the number of minterms m is a small constant multiple of the number of inputs n. Our proofs were also constructive in nature and hence one could build a Weinberger array or a gate matrix array to achieve the claimed bounds on average switching. We will free our model of the practical limitations and ask the following question. What can we say about the average switching in dynamic and static logic level structures, if m were allowed to be arbitrarily large in comparison with n? We prove the following for $m \geq n$. For a dynamic style logic, the average switching is $\Omega\,(n/2\log(\frac{m}{n}+2))$; while for the static style the lower bound is $\Omega\,(n/4\log(\frac{m}{n}+2))$. Notice that the choice between the static style and dynamic style is immaterial in this asymptotic setup. We will first show the lower bound for the dynamic case.

4.2.1 Dynamic Style

The dynamic style switching in a transition from state s_p to s_c was derived in Equation 5 on page 14.

$$Sw(s_p, s_c) = 2m - |Out(s_p)| - |Out(s_c)|$$

We need to average the expression $|Out(s_p)| + |Out(s_c)|$ over all the state pairs (s_p, s_c) and subtract that from $2m$ to derive a lower bound on the average switching. We will derive an upper bound on the average value of $|Out(s)|$ averaged over all the states s. Let us abstract the output words $Out(s)$ as $m-$bit vectors. We need to estimate the average number of 1's in an $m-$bit vector from the set $\mathcal{O}$. Since no output words are shared between input vectors, the cardinality of $\mathcal{O}$ must be 2^n.

Thus the problem is as follows. How can we construct a set $\mathcal{O}$ with 2^n $m-$bit vectors with the maximum number of 1's? We show in Appendix A that the maximum number of 1's in such a set is $\Omega(n/4\log(m/n))$ using the following argument. Let S_i denote the set of $m-$bit vectors with exactly $m-i$ 1's. The cardinality c_i of S_i is given by $\begin{pmatrix} m \\ i \end{pmatrix}$, since that is the number of ways of choosing i bit positions for zero entries. Let $\mathcal{O}$ be $\bigcup_{i=0}^{i'} S_i$ such that $\sum_{i=0}^{i'}$ is $O(2^n)$. The average number of 1's in a vector from $\mathcal{O}$ then is $O\left(\frac{\sum_i^{i'} c_i\,(m-i)}{\sum_i^{i'} c_i}\right)$. The proof of Lemma 12 in Appendix A essentially shows that the coefficient of the $n/4\log(m/n)$th term in this equation provides a tight bound on the whole sum. Hence $\frac{\sum_i^{i'} c_i\,(m-i)}{\sum_i^{i'} c_i}$ is $O(m - n/4\log([m/n]+2))$.

Lemma 9 *For $m \geq n$, the average number of 1's in an $m-$bit vector chosen at random from a set $\mathcal{O}$ of cardinality 2^n is $O(m - n/4\log([m/n] + 2))$.*

Thus the average value of $Sw(s_p, s_c)$ is $\Omega(2m - 2(m - n/4\log([m/n] + 2)))$, which is $\Omega(n/2\log([m/n]+2))$. The following corollary states that this main result which can be derived from Equation 5 and Lemma 9.

Corollary 5 *For $m \geq n$, the average switching in a dynamic style logic level structure is $\Omega(n/2\log([m/n] + 2))$.*

4.2.2 Static Style

Recall from Equation 6 on page 16 that the static style switching in a transition from state s_p to state s_c is given by the following expression.

$$Sw(s_p, s_c) = |(Out(s_p) \cap (\sim Out(s_c)))| + |((\sim Out(s_p)) \cap Out(s_c))|$$

This expression measures the Hamming distance of two $m-$bit vectors $Out(s_p)$ and $Out(s_c)$. Thus deriving a lower bound on the average switching in the static case is equivalent to deriving a lower bound on the average Hamming distance of the set $\mathcal{O}$. Notice once again that the cardinality of $\mathcal{O}$ is 2^n. In Appendix B we prove a lower bound of $n/4\log(m/n)$ on the average Hamming distance of a set S of cardinality at most 2^n containing $m-$bit vectors.

The following corollary states the main result that follows from Equation 6 and Lemma 17 in Appendix B.

Corollary 6 *For $m \geq n$, the average switching in a static style logic level structure is $\Omega\left(\frac{n}{4\log(\lceil m/n \rceil + 2))}\right)$.*

Notice that a ROM requires $m = 2^n$. In that case, by Corollary 5, the average switching for a dynamic style ROM is $\Omega(1)$. If we consider a dynamic ROM that precharges every output line to 0 through an $n-$type precharge, then it indeed has $O(1)$ average switching. From Lemma 9 the average number of 1's in an $m-$bit vector for $m = 2^n$ is given by $m - 1$. Thus there is only one zero in the output word of each state giving rise to an active-low ROM. For a static style ROM, Corollary 6 requires $\Omega(1)$ switching. Again, a static style ROM needs to switch only two output lines on average.

5 Data Path PLA

In the preceding discussion, we considered the PLAs that have the m-bit vectors with $m/2^{n-r}$ 1's as a minterm set. We also assumed that from a given state, it was equally likely to go to any other state in the next transition. Given all these assumptions, we derived some general average case results. Can we do better if we have more information about the domain of PLAs we analyze? We answer this question with regard to a very interesting and important class of PLAs, the data path control PLAs. We believe that most of the PLAs designed today are used as finite state machines (FSM).

To be able to get a handle on the structure of the minterm set M, we will make some simplifying assumptions about the behavior of the data path control FSMs. Note that the domain of our analysis is not limited to only a data path control PLA. The following assumptions hold for most other control applications too. A data path consists of many components like a shifter, ALU, register file and PC. Each such component has a cluster of control lines. When a component is logically activated, all the control lines associated with it become active over a macrocycle of the FSM. There are smaller microcycles, probably a clock cycle, which define the granularity of this cluster of control lines in more detail. This leads to the assumption that there is exactly one minterm which is associated with a cluster of control lines of one data path component. Note that this leads to an underestimation of the total switching.

For the time being, we are ignoring the state counting process, which is the heart of a FSM. Later we will refine our model, and account for the energy required by the counting. Note that we still have some inaccuracy in our model due to the clustering of all the minterms for a data path component into one minterm. With each state transition in the state counting, only a subset of the cluster of control lines for a data path component switches. However, this information is too domain specific to allow for a general model.

Observe that due to the assumptions above, this minterm set has some nice properties. Let $|\vec{u} - \vec{v}|$, the *one sided distance* of $\vec{u}$ and $\vec{v}$, be the number of bit positions where $\vec{u}$ has a one and $\vec{v}$ does not. For a pair of minterm sets $\vec{u}$ and $\vec{v}$, let the *average* one sided distances $|\vec{u} - \vec{v}| \leq k_1$ and $|\vec{v} - \vec{u}| \leq k_2$, for some constants k_1 and k_2 depending on the FSM. Let k equal $max(k_1, k_2)$. Since we have assumed that each data path unit is controlled by one *macro* control line, the number k has the following interpretation. On average, at most k new components can be activated and at most k currently active components can be deactivated in going to a next state. Note that k is, in a rough sense, the degree of parallelism or pipelining of the given data path, since on average k data path components are simultaneously active. Let us put this down as a definition.

Definition 2 *Let k be maximum of the average values of $|\vec{u} - \vec{v}|$ and $|\vec{v} - \vec{u}|$ over all the state pairs $(\vec{u}, \vec{v})$. k will be known as the* degree of parallelism *for a given data path.*

In the case of a static design, going from state s_i to state s_j, on average k minterms will be turned off and k other minterms will be turned on. Thus, the average switching will be $2k$.

On the other hand, for a dynamic design the average switching equals $2m - 2\times$*(average size of an onset)*. Assuming that k is a very good approximation to the *onset* size, the average switching here is given by $2m - 2k - 2c$ for a very small constant c.

We could define $\rho = k/m$ to be *the degree of utilization* for a FSM. Then, according to our previous discussion, a high value of ρ seems to favor the dynamic design style, while a low value of ρ favors the static design style.

State Counting: As we mentioned earlier, state counting is an essential activity associated with a FSM. Typically, there is a microprogram for activating a data path unit consisting of several microcycles. For example, the first step in using an ALU in a dual bus architecture might be latching in the two operands from two buses. The current trend in simple instruction [8] architectures seems to be to keep the number of microcycles, t, the same for each data unit. This in turn leads to uniform length machine instructions. In any case, we can assume that activating each data unit involves counting up to t, where t can be taken to be the maximum of all data unit microprogram lengths. We now analyze the switching due to counting.

There are $\lceil \log t \rceil$ output bits to count up to t. We assume the following sum of products form for the output bits of the state counter.

$$x_i(t_{n+1}) = \overline{x}_i(t_n)x_{i-1}(t_n)\ldots x_1(t_n) + x_i(t_n)\overline{x}_{i-1}(t_n) + \ldots + x_i(t_n)\overline{x}_1(t_n) \qquad (7)$$

This equation holds for all i, $1 \leq i \leq \lceil \log t \rceil$. The first product in this expression asserts the condition for toggling x_i from 0 to 1, that x_i is 0 and every other lower bit is 1. The remaining terms give conditions for retaining x_i at 1, once it is 1. Note that for the ith output bit, x_i, there are i minterms in this expression. Also notice that the set of minterms for x_i is disjoint from the set of minterms for x_j for $i \neq j$.

Note that we are limiting a FSM to lexicographic order counting. Instead, we could have worked with the Gray code counting thus switching only one bit in going to the next state. However, it does not change the relative switching of two design styles, since the number of minterms required to specify Gray code goes up by a factor of $\lceil \log t \rceil$. But we do not know how the other counting schemes will affect the switching energy of counting.

In a static design, the least significant bit, x_1, toggles t times, x_2 toggles $t/2$ times, and x_i toggles $t/2^{i-1}$ times. We state the result for a static design as a theorem.

Theorem 4 *Let a static design PLA, designed with the logic expression stated above, count up to t. The total switching in counting from* 0 *to t is given by $t \log t$.*

PROOF: We refer to the complete enumeration of $\lceil \log t \rceil$-bit vectors. Some observations about this table are as follows. Each of the $\lceil \log t \rceil$ columns has $t/2$ 0's and $t/2$ 1's. In the ith column (for $1 \le i \le \lceil \log t \rceil$), there are $t/2^i$ clusters of $2^{(i-1)}$ 1's.

We analyze the switching due to a 1-cluster in the ith column. Note that the first term in the R.H.S. of Equation 7 is *on* only for the duration of the first 1 in the cluster. At that time all the other terms for x_i are off. The $(i-1)$st column determines when the second term for x_i is on. x_i is 1 everywhere in the 1-cluster. Thus for each 0-run in the $(i-j)$th column, the $(j+1)$st term in Equation 7 stays on. Thus it is the number of 0-runs in the section of the $(i-j)$th column induced by this 1-cluster (restricted to this 1-cluster's rows), that determines the switching of the $(j+1)$st term due to this 1-cluster. Summing this over $1 \le j \le i-1$ gives the switching due to one 1-cluster in the ith column, $Switch(i) = 2 + 2 + 4 + \ldots + 2^{i-1}$. This series sums up to 2^i. Since the number of 1-clusters is $t/2^i$, the total switching due to x_i is $(t/2^i)(2^i)$, which equals t. Thus the total switching for counting is given by $t \log t$.

□

How much switching energy do we need, to count in a dynamic logic design? The answer is given by the following theorem.

Theorem 5 *Let a dynamic design PLA, designed with the logic expression in the Equation 7 above, count up to t. The total switching in counting from* 0 *to t is given by $\sim t \log^2 t$.*

PROOF: As in Theorem 4, let us estimate the switching due to one 1-cluster in the ith column. Note that the first term in the R.H.S. of Equation 7 stays *on* only until the last zero in the 0-cluster preceding a 1-cluster. Thus, for every one in a 1-cluster it is precharged and then discharged. The total switching of this term is given by $2(2^{i-1} - 1)$, where 2^{i-1} is the size of a 1-cluster. Every other term switches twice (precharge and discharge) for each zero in columns $i-1$ through 1. This switching equals $(i-1)\,2\,(2^{i-2})$. The total switching due to 1's is given by

$$\sum_{i=1}^{\lceil \log t \rceil} \frac{t}{2^i}[(2^i-2) + (i-1)2^{i-1}] .$$

This sum simplifies to the expression $t\left[\frac{\log^2 t + 3\log t}{4} - 2 + \frac{2}{t}\right] \sim t \log^2 t$. □

Let k be the degree of parallelism as defined in Definition 2, m the number of data path units and t the length of the microprogram for a data path unit. Then, the total switching with state counting is given by the following corollaries.

Corollary 7 *The average switching for a data path FSM designed in the static style is given by* $2k(t \log t)$.

Corollary 8 *The average switching for a data path FSM designed in the dynamic style is approximately* $(2m-2k)(t \log^2 t)$.

6 Delay Analysis

Based on the functional energy-time trade-off result [18], we expect a dynamic PLA to be faster than a static PLA. We show that it is indeed the case, for an *average* PLA. The following assumptions informally define, what we mean by an *average instance* of a PLA.

1. Each minterm set on average specifies $n-r$ literals and hence 2^r input values share one minterm set.

2. from a given input vector $\vec{x}$, it is equally likely to go to any other input vector $\vec{y}$, where $\vec{x}, \vec{y} \in \{0,1\}^n$.

At the risk of being redundant, we repeat the critical assumptions. Note that we are comparing the intrinsic differences between two design styles. The delay advantage gained by applying optimization techniques like folding is achievable in both the design styles. We assume that all the minterm lines have the same length in both the static and dynamic layouts, and they are routed in the same layer. In other words, all the minterms have the same capacitance and resistance. This assumption is valid for a PLA that has not gone through any layout optimizations like folding. We assume that the above mentioned conditions are valid for the output lines, as well. The techniques required to compare the output line's delay will be exactly the same as those used for comparing the minterm delay. Thus, we will demonstrate our results by minterm delay analysis.

6.1 Modified Delay Model

The VLSI complexity theory has grappled with the issue of delay modeling and each of the *synchronous, capacitive*, and *diffusion* models have been justified [2]. However, for a more exact comparison of circuits as we do here, we have to refine our model to include in our delay calculations, the *strength* and the *type* of a driver driving a node. Introducing this detail separates two commonly used design styles, static and dynamic, with respect to their speed. Although our intention is to compare these two design styles for a PLA, this model applies to any function expressed in the sum of products form. The following description justifies the need for this refinement.

Dynamic Style: An essential difference between static and dynamic design styles is that the dynamic style assigns the pull-up and pull-down phases for the same node to separate phases of a clock signal. Thus, a node is never pulled down and up, simultaneously. Let the time a minimum sized n-channel driver takes to pull down a unit area metal wire, be denoted by ν. We will assume that the time it takes a minimum sized n-channel driver to discharge a minterm wire in a given PLA instance is also given by ν, since the scale factor consisting of minterm layer capacitance and area is the same for both the styles. Thus for the sake of comparison, our results will still be valid. We will not charge any time for a $0 \rightarrow 1$ transition in a dynamic design, since the *precharge* phase can be made very very small compared to the *evaluate* phase. If it is a $1 \rightarrow 0$ transition, then the strength of this transition is defined to be the number of n-channel devices, k, pulling it down. This transition is charged a delay of ν/k. In the SPICE experiments we performed on some real PLAs, this relationship seems to hold.

Static Style: The following discussion applies to the nMOS static design style or to the pseudo-nMOS static design style. In a static design, when a node is to be pulled down to ground, there is a weak p-channel device fighting against one or more n-channel devices. The p-channel device is designed to be weak enough so that it can be overpowered even by a minimum sized n-channel device. Typically, the channel length ratio for a p-channel device to a n-channel device is about two, in a static design (Note that this ratio is required in order to have reasonable noise immunity, as shown in the text book by Weste and Eshraghian [20] [page 54]). In the SPICE experiments conducted by the author, the time a n-channel device took to discharge a node without a weak p-channel device pulling it up, was almost the same as the time in a static setup. Thus, we assume that a minimum sized n-type device takes time ν to pull down a minterm node even in the static design style. The time it takes for a p-channel device to pull up the same node is different due to lower mobility of a p-channel. Let μ_r be the ratio of n-channel mobility to p-channel mobility, *i.e.*, $\mu_r = \mu_n/\mu_p$. Let us denote the time, $\mu_r\nu$, that a minimum sized p-channel transistor takes to pull up a minterm, by ν_p. Then a p-channel device with a channel width ratio of r, will take time $r\nu_p$. This case, when a weak p-type pulls up a node, shows the main difference between the two design styles. As we previously mentioned, this time equals $r\mu_r\nu$, where r is approximately two and μ_r in a typical process today varies between two and three. Thus a $0 \to 1$ transition takes ~ 5 times as long as the slowest $1 \to 0$ transition (strength 1, when only one n-channel device is on).

To summarize the delay estimation, $0 \to 1$ transitions are expensive in a static design, while they come for free in a dynamic design. Each $0 \to 1$ transition takes $\sim 5\nu$ delay, while a strength[6] k $1 \to 0$ transition costs only ν/k. In the next section, we show that an average PLA instance has many $0 \to 1$ transitions, hence proving our claim.

6.2 Dynamic vs Static PLA Delay

Theorem 6 *The asymptotic average time to switch a static PLA is ~ 5 times as long as the average time to switch a dynamic PLA.*

PROOF: Let us again visualize a lexicographically ordered enumeration of 2^{n-r} $m-$bit vectors in $\mathcal{M}$. We wish to determine the average number of $0 \longrightarrow 1$ switchings per transition. This is essentially the number of positions where s_p contains a 0 and s_c contains a 1. It seems somewhat like a *one*

[6]Recall that a transition has strength k if k devices are simultaneously changing the state of the same node in the same direction.

sided Hamming distance, $h_o(\vec{u}, \vec{v})$, where we count only the positions with a 0 in $\vec{u}$ and a 1 in $\vec{v}$. By Lemma 3, this average one sided Hamming distance is given by $\frac{(2^{n-r}-1)}{2^{2(n-r)}}m$. This multiplied by $\sim 5\nu$ gives the average static evaluation time, while a factor of ν estimates the dynamic evaluation time. Hence the asymptotic average time to switch a static PLA is $\sim$ 5 times as long as the average time to switch a dynamic PLA.

$\square$

Note that in this delay model, the superiority of a dynamic design over a static design can be proved for any logic expression by showing that a majority of transitions are from zero to one. In practice, we have the same clock period for each transition. Hence we do not have to show that almost all the transitions are the zero to one kind. Even if one transition has a $0 \longrightarrow 1$ switching component, from practical considerations the clock period will have to be long enough to accommodate it.

7 Conclusions

We raised the question, "How do dynamic and static design styles compare in terms of energy consumption and speed?". For a PLA implementation, the answer depends on two factors, r – the average number of *don't cares* per minterm and s – the average number of minterms per output bit. Corollaries 3 and 4 provide simple expressions to determine this. A silicon compiler presumably could use these expressions to make a choice between two desin styles.

The result for general logic level structures is more clear-cut. The average switching in the dynamic case is about twice the average switching in the static case. On the other hand, as the number of output bits m grows asymptotically larger than the number of input bits n (as in a ROM), both the styles consume the same order of energy. The expected delay of a dynamic style logic, however, is approximately 5 times smaller than that of static style logic.

There are several related open questions that we did not address in this paper. We still have not compared different CMOS design styles such as Domino, NORA, CVSL etc. for their relative energy consumption. Given a function of n input bits and m output bits and its domino implementation with k domino stages, it is hard to determine a reasonable distribution of these input and output bits among k stages. Recently there has been some activity in the area of decomposition and factorization of finite state machines [5]. Can our analysis be carried over to this kind of FSM synthesis and used as an optimization criterion?

References

[1] A. Aggarwal, A. K. Chandra, and P. Raghavan. Energy Consumption in VLSI Circuits. In *ACM Symposium on Theory of Computing*, pages 205–216, ACM-SIGACT, 1988.

[2] G. Bilardi, M. Pracchi, and F. P. Preparata. A Critique of Network Speed in VLSI Models of Computation. *IEEE J. of Solid-State Circuits*, August 1982.

[3] R. Brayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis.* Kluwer Academic Publishers, Boston, Mass., 1984.

[4] G. De Micheli and A. Sangiovanni-Vincentelli. *PLEASURE: A Computer Program for Simple/Multiple Constrained/Unconstrained Folding of Programmable Logic Arrays.* Memorandum No. UCB/ERL M82/57, U.C. Berkeley, 1982.

[5] S. Devadas and A.R. Newton. Decomposition and Factorization of Sequential Finite State Machines. In *Proceedings of Sixth IEEE International Conference on Computer-Aided Design*, pages 148–151, 1988.

[6] G. Hachtel, A.R. Newton, and A. Sangiovanni-Vincentelli. An Algorithm for Optimal PLA Folding. *IEEE Trans. on CAD of ICs and Systems*, April 1982.

[7] L. G. Heller, W. R. Griffin, J. W. Davis, and N. G. Thoma. Cascade Voltage Switch Logic: A Differential CMOS Logic Family. In *Proceedings of the IEEE International Solid state Circuits Conference*, IEEE, February 1984.

[8] S. Ho, B. Jinks, T. Knight, J. Schaad, L. Snyder, A. Tyagi, and C. Yang. The Quarter Horse: A Case Study in Rapid Prototyping of a 32-bit Microprocessor Chip. In *IEEE Proceedings of the International Conference on Computer Design: VLSI in Computer*, IEEE Computer Society, 1985.

[9] G. Kissin. Functional Bounds on Switching Theory. In *Chapel Hill Conference on VLSI*, U.N.C., Chapel Hill, Computer Science Press, 1985.

[10] G. Kissin. *Measuring Energy Consumption in VLSI Circuits.* PhD thesis, Department of Computer Science, University of Toronto, Toronto, 1987.

[11] G. Kissin. Measuring Energy Consumption in VLSI Circuits: a Foundation. In *ACM Symposium on Theory of Computing*, ACM-SIGACT, 1982.

[12] A. D. Lopez and H. S. Law. A Dense Gate Matrix Layout Method for MOS VLSI. *IEEE Transactions on Electron Devices*, ED-27:8, August 1980.

[13] C. Lutz. *Design of the Mosaic Processor.* Masters Thesis, Computer Science Dept., California Institute of Technology, 1984.

[14] C. Mead and L. Conway. *Introduction to VLSI Systems.* Addison-Wesley, Reading, Mass., 1980.

[15] W.D. Moeller and G. Sandweg. The Peripheral Processor PP4: A Highly Regular VLSI Processor. In *Proceedings of 11th International Symposium on Computer Architecture*, 1984.

[16] L. Snyder and A. Tyagi. *The Energy Complexity of Transitive Functions.* Technical Report TRCS-86-09-07, Dept. of Computer Science, University of Washington, Seattle, 1986.

[17] G. B. Thomas. *Calculus and Analytic Geometry.* Addison-Wesley, Reading, Mass., fourth edition, 1969.

[18] A. Tyagi. *The Role of Energy in VLSI Computations.* PhD thesis, Department of Computer Science, University of Washington, Seattle, 1988. Available as UWCS Technical Report Number 88-06-05.

[19] A. Weinberger. Large Scale Integration of MOS Complex Logic : A Layout Method. *IEEE Journal of Solid State Circuits*, SC-2:4:182–190, December 1967.

[20] N. Weste and K. Eshraghian. *Principles of CMOS VLSI Design.* Addison-Wesley, Reading, Mass., 1985.

Appendix

A Upper Bound on Average Weight

Let S' be the set of 2^m $m-$bit strings. We wish to build a set S of at least 2^n of these strings that contains maximum average number of 1's per string. We assemble a set $S \subseteq S'$ with cardinality not exceeding 2^n that has the maximum average weight for a set of its size. The average weight of S definitely constitutes an upper bound on the weight of any set containing 2^n m-bit strings. How is the set S built? Let S_i (of cardinality c_i) be the set of all the strings in S' with $m - i$ 1's. The greedy algorithm below constructs the set S. c_S denotes the cardinality of set S.

```
$S = \emptyset$;  $c_S = 0$;  $i = 0$;
**while** $(c_S + c_i) \leq 2^n$ **do**
    $S = S \cup S_i$;  $c_S = c_S + c_i$;  $i = i + 1$;
**endwhile**
```

Let i'' be the index of the set S_i that was last included in S, *i.e.*, this algorithm exits with $i = i'' + 1$. The problem is to find an approximation i' to i'' such that $\sum_{i=0}^{i'} c_i \leq 2^n$. Then the average weight of a set containing atleast 2^n m-bit strings $W(m,n)$ is given by

$$W(m,n) = O\left(\frac{\sum_i^{i'} c_i\,(m-i)}{\sum_i^{i'} c_i}\right) \tag{8}$$

Let us first estimate c_i, the number of all the $m-$bit strings with weight $m - i$. c_i is just the number of ways of placing i 0's in m positions. This is given by $\begin{pmatrix} m \\ i \end{pmatrix}$.

Our strategy to estimate a lower bound in Equation 8 is as follows. We first show in Lemma 10 that for an integer m, in the sequence $\left\{\begin{pmatrix} m \\ 0 \end{pmatrix}, \begin{pmatrix} m \\ 1 \end{pmatrix}, \begin{pmatrix} m \\ 2 \end{pmatrix}, \ldots, \begin{pmatrix} m \\ m \end{pmatrix}\right\}$ the term $\begin{pmatrix} m \\ j \end{pmatrix}$ is at least as large as $\sum_{i=0}^{j-1} \begin{pmatrix} m \\ i \end{pmatrix}$ for $j \leq m/3$. This fact helps in the following way. We wish to derive a lower bound on $W(m,n)$ by deriving a lower bound on the expression $\dfrac{\sum_i^{i'} i \begin{pmatrix} m \\ i \end{pmatrix}}{\sum_i^{i'} \begin{pmatrix} m \\ i \end{pmatrix}}$.

If $\dfrac{\begin{pmatrix} m \\ i' \end{pmatrix}}{\sum_i^{i'} \begin{pmatrix} m \\ i \end{pmatrix}} \geq 1/2$ then $\Omega(i')$ provides a lower bound on $W(m,n)$. Hence the value of i' we choose later must satisfy the relation $i' \leq m/3$.

Before we can prove Lemma 10, let us develop some notation. Let $s_{m,j}$ denote $\sum_{i=0}^{j} \begin{pmatrix} m \\ i \end{pmatrix}$. Let $b_{m,i}$ denote $\begin{pmatrix} m \\ i \end{pmatrix}$. Then the following lemma holds.

Lemma 10 *In the sequence* $\{b_{m,i}\}_{i=0}^{m}$, $b_{m,j} \geq s_{m,j-1}$ *for* $j \leq m/3$.

Proof: Note that for $j \leq m/3$, $b_{m,j}/b_{m,j-1} \geq 2$. To see it, let us consider the ratio $b_{m,j}/b_{m,j-1} = \frac{m-j+1}{j}$. This ratio is larger than 2 for $j \leq m/3$. The remaining proof is by induction.

inductive hypothesis: $b_{m,j} \geq s_{m,j-1}$ for $j \leq m/3$.

basis: For $j = 1$ (assume $m > 3$), $s_{m,0} = b_{m,0} = 1$ and $b_{m,1} = m$.

inductive step: Let the inductive hypothesis hold for all $j < l \leq m/3$. Consider $s_{m,l-1} = s_{m,l-2} + b_{m,l-1} \leq^{\text{by inductive hypothesis}} b_{m,l-1} + b_{m,l-1}$. And since $2b_{m,l-1} \leq b_{m,l}$, it shows that $s_{m,l-1} \leq b_{m,l}$.

□

Now we claim that $\frac{k}{4\log(m/n)}$ is a valid choice for i'. Notice first that for $m \geq 2n$, $\frac{n}{4\log(m/n)} \leq m/3$. Another condition that i' must satisfy is $\sum_{i=0}^{i'} c_i \leq 2^n$. It turns out that due to Lemma 10 and the fact that $i' \leq m/3$, this condition reduces to the condition that $c_{i'} = O(2^n)$. If $c_{i'} = \begin{pmatrix} m \\ i' \end{pmatrix}$ is $O(2^n)$ then by Lemma 10 $\sum_{i=0}^{i'} c_i$ is also $O(2^n)$. But we are looking for an i' such that

$\sum_{i=0}^{i'} c_i \leq 2^n$. Let $c_{i'}$ be $O(2^n)$ with a proportionality constant of K. By Lemma 10, $c_{i'-1}$ will be at most $\frac{K}{2}2^n$. In general, $c_{i'-j}$ will be at most $\frac{K}{2^j}2^n$. Hence $c_{(i'-\log K)} \leq 2^n$. Since we are only looking for a lower bound on i' and $(i'-\log K)$ is $\Omega(i')$, all that we require of i' is that $c_{i'}$ be $O(2^n)$. We claim that $c_{n/4\log((m/n)+2))}$ is $O(2^n)$. We only need to show that $\binom{m}{n/4\log[(m/n)+2])}$ is $O(2^n)$. We state and prove this claim as a lemma.

Lemma 11 *Let m and n be two positive integers such that $m \geq n$. Then $\binom{m}{\frac{n}{4\log([m/n]+2)}}$ is $O(2^n)$.*

Proof: We consider two cases, one: when $m \geq n^2$, and two: when $n \leq m \leq n^2$.

$m \geq n^2$: Let us consider $\binom{m}{i}$. It equals $\frac{m(m-1)...(m-i+1)}{i(i-1)...1}$. For non-negative m and i, the largest fraction among $\frac{m}{i}, \frac{m-1}{i-1}, \ldots, \frac{m-i+1}{1}$ is $\frac{m-i+1}{1} \leq m$. Thus an upper bound on $\binom{m}{i}$ is m^i. For $i = \frac{n}{4\log(m/n)}$, this upper bound becomes $m^{\frac{n}{4\log(m/n)}}$. This equals $(m^{n/\log m})^{\frac{\log m}{4\log(m/n)}}$. $m^{n/\log m}$ is 2^n. For $m \geq n^2$, $\frac{\log m}{4\log(m/n)} < 1$. Hence $\binom{m}{\frac{n}{4\log(m/n)}} \leq 2^n$. Note that $\binom{m}{\frac{n}{4\log([m/n]+2)}} \leq \binom{m}{\frac{n}{4\log(m/n)}}$ since $\frac{n}{4\log(m/n)} \leq m/2$ for the range of interest.

$n \leq m \leq n^2$: Stirling's approximation for $k!$ is given by $\sqrt{2\pi k}(\frac{k}{e})^k$, where e is the inverse natural log of one. This is a tight bound in asymptotics. Then $\binom{m}{i}$ is $\Theta\left(\frac{\sqrt{2\pi m}(m/e)^m}{\sqrt{2\pi i}(i/e)^i\sqrt{2\pi(m-i)}((m-i)/e)^{m-i}}\right)$. This equals $\frac{1}{\sqrt{2\pi}}\sqrt{\frac{m}{i(m-i)}}\left(\frac{m}{i}\right)^i\left(\frac{m}{m-i}\right)^{m-i}$. Note that the last term in this expression, $\left(\frac{m}{m-i}\right)^{m-i}$ equals $\left(1+\frac{i}{m-i}\right)^{m-i}$. We claim that this expression has an upper bound of e^i. Let us prove it.

Claim: $\left(1+\frac{i}{m-i}\right)^{m-i} \leq e^i$.

Proof of the Claim: $(1+x)^{1/x}$ is $e^{\frac{\ln(1+x)}{x}}$. It is well known from any calculus text book [15] that $\lim_{x\to 0}(1+x)^{1/x} = e$. This requires taking the limit of the exponent of e in this expression, $\frac{\ln(1+x)}{x}$, using l'Hôpital's rule. The exponent of e, $\frac{\ln(1+x)}{x}$, goes to 1 in the limit as $x \to 0$. As x increases, this exponent can only decrease since $\ln(1+x)$ has a slower growth rate than x. Another way to see that $(1+x)^{1/x}$ is a decreasing function of x is as follows. The derivative of $(1+x)^{1/x}$ with respect to x is given by $(1+x)^{1/x}\left(\frac{1}{x^2+x} - \frac{\ln(1+x)}{x^2}\right)$. This expression is always negative for $x > 0$. Thus a value of x greater than 0 only makes this expression smaller than e. This implies that $(1+x)^{1/x}$ is e^c for a constant $c \leq 1$.

Now notice that $\left(1+\frac{i}{m-i}\right)^{m-i}$ is equivalent to $\left[(1+x)^{(1/x)}\right]^i$ with $x = \frac{i}{(m-i)}$. Notice that for $m > k$, $x > 0$. Since $(1+x)^{1/x} = e^c$ for a $c \leq 1$, $\left(1+\frac{i}{m-i}\right)^{m-i} \leq e^i$.

End of Claim

For $i = \frac{n}{4\log([m/n]+2)}$, an upper bound on $\begin{pmatrix} m \\ i \end{pmatrix}$ is given by the following expression.

$$\frac{1}{\sqrt{2\pi}}\sqrt{\frac{m}{i(m-i)}}\;\overbrace{\left(\frac{m}{i}\right)^i}^{(A)}\;\overbrace{e^i}^{(B)}$$

The factor $\sqrt{\frac{m}{2\pi i(m-i)}}$ is $\sqrt{\frac{2\log([m/n]+2)\,m}{\pi n\left(m-\frac{n}{4\log([m/n]+2)}\right)}}$. In asymptotics, when $m >> n$, this term is $O(1/\sqrt{n})$. Even for $m = 3k$ though, this term equals $\sqrt{\frac{\log 3}{\pi\left(n-\frac{n}{4\log 3}\right)}}$. Hence this square root term is $O(1)$ for $n < m \leq n^2$. Now let us consider Term (A). $(\frac{m}{i})^i$ is given by $(\frac{m}{n}4\log([m/n]+2))^{(n/4\log([m/n]+2))}$. This has an upper bound of $(m/n)^{\frac{n/4}{\log(m/n)}}(4\log([m/n]+2))^{\frac{3n/4}{3\log([m/n]+2)}}$ which in turn is $2^{(n/4)}\,2^{\left((3n/4)\,\frac{2+\log\log([m/n]+2)}{3\log([m/n]+2)}\right)}$.

Now consider Term (B). e^i equals $2^{i\log e}$. This is given by $2^{(n\ \log e)/(4\log([m/n]+2))}$ or alternatively by $2^{\left((3n/4)\,\frac{\log e}{3\log([m/n]+2)}\right)}$. Terms (A) and (B) combined together give $2^{(n/4)}$ $2^{\left((3n/4)\,\left[\frac{\log e+2+\log\log([m/n]+2)}{3\log([m/n]+2)}\right]\right)}$. For $m \geq n$, the fraction $\frac{\log e+2+\log\log([m/n]+2)}{3\log([m/n]+2)}$ evaluates to a value less than 1. In particular, for $m = n$ it evaluates to .8637. Hence for $n^2 > m \geq n$, $\begin{pmatrix} m \\ \frac{n}{4\log([m/n]+2)} \end{pmatrix}$ is $O(2^n)$.

$\square$

Now we show that Lemma 11 indeed leads to a good lower bound on the average weight $W(m,n)$.

Lemma 12 *Let m and n be two positive integers such that $m \geq n$. The average weight of a set S containing at least $m-$bit vectors, $W(m,n) = O(m - n/4\log([m/n]+2))$.*

PROOF: Notice that for $m \geq n$, $n/4\log([m/n]+2) \leq m/3$. Equation 8 gives an upper bound of $\Omega\left(\frac{\sum_{i=0}^{i'} c_i\, i}{\sum_{i=0}^{i'} c_i}\right)$ on $W(m,n)$, provided $\sum_i^{i'} \begin{pmatrix} m \\ i \end{pmatrix} \leq 2^n$. Let us use $n/4\log([m/n]+2)$ for i'. Lemma 11 shows that for $m \geq n$ $\begin{pmatrix} m \\ \frac{n}{4\log([m/n]+2)} \end{pmatrix}$ is $O(2^n)$. Hence $c_{i'} = \begin{pmatrix} m \\ \frac{n}{4\log([m/n]+2)} \end{pmatrix}$ is $O(2^n)$. Lemma 10 states that since $n/4\log([m/n]+2) \leq m/3$, $\sum_{i=0}^{\left(\frac{n}{4\log([m/n]+2)}-1\right)} c_i \leq c_{n/4\log([m/n]+2)}$. Thus $\sum_{i=0}^{\left(\frac{n}{4\log([m/n]+2)}\right)} c_i \leq 2c_{n/4\log([m/n]+2)}$ and hence is $O(2^n)$.

Now let us estimate a lower bound on $\frac{\sum_{i=0}^{n/4\log([m/n]+2)} c_i\,(m-i)}{\sum_i^{n/4\log([m/n]+2)} c_i}$. We noted earlier that $\sum_i^{\frac{n}{4\log([m/n]+2)}} c_i$ is $\Theta(c_{n/4\log([m/n]+2)})$. Hence Equation 8 translates into $W(m,n) = O(m - n/4\log([m/n]+2))$.

□

B Lower Bound on Average Hamming Distance

Before we prove a lower bound on $h_a(\mathcal{O})$, let us prove a lemma analogous to Lemma 10. Once again, let $b_{m,i}$ denote $\begin{pmatrix} m \\ i \end{pmatrix}$. Let $s'_{m,j}$ denote $\sum_{i=0}^{j} \begin{pmatrix} m \\ i \end{pmatrix}\begin{pmatrix} m \\ i \end{pmatrix}$.

Lemma 13 *In the sequence $\{b^2_{m,i}\}_{i=0}^{m}$, $b^2_{m,j} \geq s'_{m,j-1}$ for $j \leq m/3$.*

PROOF: Note that for $j \leq m/3 < m/(1+\sqrt{2})$, $b^2_{m,j}/b^2_{m,j-1} \geq 2$. To see it, let us consider the ratio $b_j^2/b_{j-1}^2 = \frac{(m-j+1)^2}{j^2}$. This ratio is larger than 2 for $j \leq m/(1+\sqrt{2})$. The remaining proof is by induction and is identical to the proof of Lemma 10.

□

The strategy to derive a lower bound on $h_a(\mathcal{O})$ is as follows. We will construct a set S of $m-$bit vectors that has the minimum average Hamming distance for its size. We also show that adding any more strings to S will only increase the average Hamming distance within the range of interest. Thus any set S' with $|S'| \geq |S|$ satisfies the condition, $h_a(S') \geq h_a(S)$. Note that $|\mathcal{O}| = 2^n$. Hence a set S of size no more than 2^n will provide a lower bound on the average Hamming distance of $\mathcal{O}$. Thus $h_a(S)$ is a lower bound on $h_a(\mathcal{O})$ for a carefully selected set S. Let us first describe how this set S is constructed.

Let $S_{m,i}$ denote the set of all the $m-$bit strings with exactly i 1's. Thus $S_{2,0}$ contains only 00 and $S_{2,1}$ contains 01 and 10. There are $\binom{m}{i}$ strings in $S_{m,i}$. We claim that a set of the form $S_k = \cup_{i=0}^{k} S_{m,i}$, for $k \leq m/2$, has the smallest average Hamming distance for its size. As we saw in Appendix A, for $k = n/4\log(m/n)$, $|\cup_{i=0}^{k} S_{m,i}| \leq 2^n$. Thus for the range of our interest, $S_{n/4\log(m/n)}$ can be chosen for S and it will indeed be a set of minimum average Hamming distance for its size.

Let d_k denote the average Hamming distance of the set $S_{m,k}$. Also let $d_{j,k}$ denote the average Hamming distance between the sets $S_{m,j}$ and $S_{m,k}$. (*The average Hamming distance between two sets X and Y is given by* $\frac{\sum_{\bar{x}\in X,\bar{y}\in Y} h(\bar{x},\bar{y})}{|X||Y|}$.) Let us first derive the values of d_k and $d_{i,i+j}$.

Lemma 14 *The average Hamming distance of the set $S_{m,k}$ as defined above, d_k, is given by $2k(m-k)/m$. The average Hamming distance between two sets $S_{m,i}$ and $S_{m,i+j}$, denoted by $d_{i,i+j}$ is $i\frac{m-i-j}{m} + (i+j)\frac{m-i}{m}$.*

Proof: Let us first estimate d_k. Note that $|S_{m,k}| = \binom{m}{k}$. Consider an enumeration of all the vectors in $S_{m,k}$ with one vector per row and with the lth bit of each vector in the lth column for $1 \leq l \leq m$. Each column in this enumeration contains $\binom{m-1}{k-1}$ 1's, since that is the number of ways of placing the remaining $k-1$ 1's in the other $m-1$ columns. Then the number of zeros in each column is $\binom{m}{k} - \binom{m-1}{k-1} = \binom{m-1}{k}$. For each column, the rows containing $\binom{m-1}{k-1}$ 1's contribute $\binom{m-1}{k-1}\binom{m-1}{k}$ towards the total Hamming distance. Similarly the rows containing zeros in that column contribute the same amount to the total Hamming distance. Thus the total Hamming distance of $S_{m,k}$ is $2m\binom{m-1}{k-1}\binom{m-1}{k}$.

Then d_k is $\frac{2m\binom{m-1}{k-1}\binom{m-1}{k}}{\binom{m}{k}\binom{m}{k}}$, which equals $2k(m-k)/m$. For $1 \leq k \leq (m-1)/2$, d_k is $\Theta(k)$.

Now let us calculate $d_{i,i+j}$ for $j > 0$. This will give us a general way of evaluating $d_{l,k}$ for $1 \leq l \leq k-1$. Once again, in an enumeration of $S_{m,i+j}$ each column contains $\binom{m-1}{i+j-1}$ 1's and $\binom{m-1}{i+j}$ 0's. Similarly, each column in an enumeration of $S_{m,i}$ has $\binom{m-1}{i-1}$ 1's and $\binom{m-1}{i}$ 0's. Thus the interaction of one column of $S_{m,i+j}$ and $S_{m,i}$ provides $\binom{m-1}{i+j-1}\binom{m-1}{i} + \binom{m-1}{i-1}\binom{m-1}{i+j}$ towards total Hamming distance. Thus the average Hamming distance $h_a(S_{m,i}, S_{m,i+j})$ is $m\dfrac{\binom{m-1}{i+j-1}\binom{m-1}{i}+\binom{m-1}{i-1}\binom{m-1}{i+j}}{\binom{m}{i}\binom{m}{i+j}}$, which is $i\frac{m-i-j}{m} + (i+j)\frac{m-i}{m}$.

□

Now let us prove our claim about the set S_k being the right kind of set to derive a lower bound on the average Hamming distance.

Lemma 15 *The set $S_k = \cup_{i=0}^{k} S_{m,i}$, for $k \leq (m-1)/2$ and $S_{m,i}$ as described above, has the minimum average Hamming distance of a set of its cardinality.*

PROOF: The proof is by induction on k.

inductive hypothesis: For $S_j = \cup_{i=0}^{j} S_{m,i}$, for $j \leq (m-1)/2$, $h_a(S_j) \leq h_a(S)$ for any set $S \subseteq \{0,1\}^m$ with $|S| \geq |S_j|$.

basis: Consider the case $j = 0$. The set $S_0 = \{000\cdots00\}$. This set has an average Hamming distance of 0 and no other set of cardinality 1 has a lower average Hamming distance.

inductive step: Let the claim be true for all $j < k$. Let us consider the set $S_k = \cup_{i=0}^{k} S_{m,i}$. We have to prove two things. One, given that we are building on top of the set S_{k-1}, $S_{m,k}$ is the best set to add; two, there is no other set of cardinality $|S_k|$ that has a lower average Hamming distance.

Let us prove the first claim first. Consider a set S', a small perturbation on $S_{m,k}$, to extend S_{k-1} by $|S_{m,k}|$ elements. The total Hamming distance of $S_{k-1} \cup S'$, $h_t(S_{k-1} \cup S')$ is $h_t(S_{k-1}) + h_t(S') + h_t(S_{k-1}, S')$. On the other hand, $h_t(S_k) = h_t(S_{k-1}) + h_t(S_{m,k}) + h_t(S_{k-1}, S_{m,k})$. Let S' contain one vector $\vec{x}'$ with $k+1$ 1's, where $\vec{x}'$ is essentially derived by adding a 1 to a vector $\vec{x} \in S_{m,k}$ in the lth bit position. This is the smallest perturbation on $S_{m,k}$. Let us estimate $h_t(S')$. Note that the lth bit of $\vec{x}$ is 0 and the lth bit of $\vec{x}'$ is 1. Let us concentrate on the lth column of an enumeration of S'. It has $\begin{pmatrix} m-1 \\ k-1 \end{pmatrix} + 1$ 1's and $\begin{pmatrix} m-1 \\ k \end{pmatrix} - 1$ 0's. Thus the Hamming distance contribution of this column is $2\left[\left(\begin{pmatrix} m-1 \\ k-1 \end{pmatrix} + 1\right)\left(\begin{pmatrix} m-1 \\ k \end{pmatrix} - 1\right)\right]$. This equals $2\begin{pmatrix} m-1 \\ k-1 \end{pmatrix}\begin{pmatrix} m-1 \\ k \end{pmatrix} + 2\left(\begin{pmatrix} m-1 \\ k \end{pmatrix} - \begin{pmatrix} m-1 \\ k-1 \end{pmatrix} - 1\right)$. The contribution due to this extra 1, $2\left(\begin{pmatrix} m-1 \\ k \end{pmatrix} - \begin{pmatrix} m-1 \\ k-1 \end{pmatrix} - 1\right) \geq 0$, since for $k \leq (m-1)/2$, $\begin{pmatrix} m-1 \\ k \end{pmatrix}$ is a monotonically increasing function of k. Thus the total Hamming distance of S', $h_t(S') \geq h_t(S_{m,k})$.

We can similarly compare $h_t(S_{k-1}, S_{m,k})$ with $h_t(S_{k-1}, S')$. As we observed earlier, each column in $S_{m,i}$ for $i < k$ contains $\begin{pmatrix} m-1 \\ i-1 \end{pmatrix}$ 1's and $\begin{pmatrix} m-1 \\ i \end{pmatrix}$ 0's. Let us consider the Hamming distance of the vector $\vec{x}'$ with $S_{m,i}$. $h_t(\vec{x}', S_{m,i}) = h_t(\vec{x}, S_{m,i}) + \begin{pmatrix} m-1 \\ i \end{pmatrix} - \begin{pmatrix} m-1 \\ i-1 \end{pmatrix}$, since the lth column contains one extra one and one fewer zeros. The extra term is definitely positive for $i \leq (m-1)/2$. Thus $h_t(\vec{x}', S_{m,i}) > h_t(\vec{x}, S_{m,i})$ for $i \leq (m-1)/2$. Just by adding an extra one to $S_{m,k}$, we see that the average Hamming distance goes up. By a simple induction on the number of ones added to $S_{m,k}$, we can show that adding $S_{m,k}$ to S_{k-1} indeed gives a set with the least Hamming distance over all ways to produce it.

Now let us prove the second claim. Let S' be a set of cardinality $|S_k|$. Let us consider the following partition of S'. Let $T_l \subset S'$ have the smallest average Hamming distance of all the $|S_{k-1}|$ sized subsets of S'. Note that $|T_l|$ also is $|S_{k-1}|$. Let T_u be the set difference of S' and T_l, *i.e.* $T_u = S' - T_l$. Then $S' = T_l \cup T_u$. The expression for the total Hamming distance of S' is given by $h_t(S') = h_t(T_l) + h_t(T_u) + h_t(T_l, T_u)$. By inductive hypothesis, $h_t(T_l) \geq h_t(S_{k-1})$. If $T_l = S_{k-1}$, then we are dealing with the case described above, so let us assume that $T_l \neq S_{k-1}$. T_u cannot have any vectors from S_{k-1}. Hence by Lemma 14 $h_t(T_u) \geq h_t(S_{m,k})$. Now we claim that $h_t(T_l, T_u)$ is also at least as high as $h_t(S_{k-1}, S_{m,k})$. We wish to show that $h_t(A, B) \geq h_t(A, B')$ for a set A if $h_t(B) \geq h_t(B')$. Let the average number of ones (zeros) in a column of a set S be denoted by p_S (q_S). Then $h_t(S) = 2mp_Sq_S$, where $q_S = |S| - p_S$. This achieves a maximum when $q_S = p_S$. Notice that $h_t(B) \geq h_t(B')$ implies that $ABS(p_B - q_B) \geq ABS(p_{B'} - q_{B'})$. It can be seen that $p_Aq_B + q_Ap_B \geq p_Aq_{B'} + q_Ap_{B'}$. Hence $h_t(T_l, T_u) \geq h_t(S_{k-1}, T_u)$. We already showed in the first part of this proof that $h_t(S_{k-1}, S) \geq h_t(S_{m,k})$ for any set S with larger number of 1's than $S_{m,k}$ and cardinality $|S_{m'k}|$. Since we know that T_u contains at least as many ones as $S_{m,k}$, and its cardinality is $|S_{m,k}|$; $h_t(S') \geq h_t(S_k)$.

Thus we have shown that the total Hamming distance of S_k, $h_t(S_k)$, is one of the smallest for a set of size $|S_k|$. Hence the average Hamming distance of S_k, $h_a(S_k)$, also is minimum for $k \leq (m-1)/2$. In addition, the average contributions $d_{i,i+j}$ and d_l are monotonically increasing functions of j and l respectively for $j > 0$ and $i, l \leq (m-1)/2$. Thus we have also shown that increasing the size of S_k can only increase the average Hamming distance.

□

In order for $|S_k|$ to be $O(2^n)$, $\sum_{i=0}^{k} \begin{pmatrix} m \\ i \end{pmatrix}$ must be $O(2^n)$. We have seen in Appendix A, Lemma 11 that for $m \geq n$, $k = n/4 \log(m/n)$ satisfies this condition. Now to derive a lower bound on $h_a(\mathcal{O})$, all that needs to be done is to derive a lower bound on $h_a(S_{n/4\log(m/n)})$. The following lemma does that.

Lemma 16 *The average Hamming distance of the set S_k, $h_a(S_k)$, for $k \leq m/3$ is $\Omega(k)$.*

PROOF: The average Hamming distance of S_k is given by the following equation.

$$h_a(S_k) = \frac{\left[\sum_{j=0}^{k} h_a(S_{j-1}, S_{m,j}) \left(\begin{pmatrix} m \\ j \end{pmatrix} \sum_{i=0}^{j-1} \begin{pmatrix} m \\ i \end{pmatrix} \right) \right] + \left[\sum_{i=0}^{k} h_a(S_{m,i}) \begin{pmatrix} m \\ i \end{pmatrix}^2 \right]}{\left[\sum_{j=0}^{k} \left(\begin{pmatrix} m \\ j \end{pmatrix} \sum_{i=0}^{j-1} \begin{pmatrix} m \\ i \end{pmatrix} \right) \right] + \sum_{i=0}^{k} \begin{pmatrix} m \\ i \end{pmatrix}^2}$$

Consider the denominator in this equation. The first term $\sum_{j=0}^{k}\left(\begin{pmatrix} m \\ j \end{pmatrix}\sum_{i=0}^{j-1}\begin{pmatrix} m \\ i \end{pmatrix}\right)$ is at most as large as the second term $\sum_{i=0}^{k}\begin{pmatrix} m \\ i \end{pmatrix}\begin{pmatrix} m \\ i \end{pmatrix}$ by Lemma 10 for $k \leq m/3$. Thus $h_a(S_k) \geq \dfrac{\sum_{i=0}^{k} h_a(S_{m,i})\begin{pmatrix} m \\ i \end{pmatrix}^2}{2\sum_{i=0}^{k}\begin{pmatrix} m \\ i \end{pmatrix}^2}$ for $k \leq m/3$. Lemma 13 asserts that for $k \leq m/3$, $\dfrac{\begin{pmatrix} m \\ k \end{pmatrix}^2}{\sum_{i=0}^{k}\begin{pmatrix} m \\ i \end{pmatrix}^2} \geq 1/2$. Then, $h_a(S_k)$ is $\Omega(h_a(S_{m,k}))$. From Lemma 14, $h_a(S_{m,k})$ equals $2k(m-k)/m$, which for $k \leq m/3$ is $\Theta(k)$. Thus $h_a(S_k)$ is $\Omega(k)$ for $k \leq m/3$.

□

The next lemma places a lower bound on the average Hamming distance of the set $\mathcal{O}$.

Lemma 17 *The average Hamming distance of the set $\mathcal{O}$ is $\Omega(n/\log(m/n))$ for $m \geq n$.*

PROOF: Let us consider the set $S_{n/4\log(m/n)} = \bigcup_{i=0}^{n/4\log(m/n)} S_{m,i}$. Its cardinality is no greater than $O(2^n)$ for $m \geq n$. It has the minimum average Hamming distance of a set of its size by Lemma 15. Since the cardinality of $\mathcal{O}$ is 2^n, $h_a(\mathcal{O})$ is $\Omega(h_a(S_{n/4\log(m/n)}))$. From Lemma 16, $h_a(S_{n/4\log(m/n)}))$ is $\Omega(n/4\log(m/n))$ for $n/\log(m/n) \leq m/3$. And hence that is also a lower bound on $h_a(\mathcal{O})$ for $m \geq n$.

□

ACKNOWLEDGEMENTS
I am thankful to Carl Ebeling and Larry Snyder of University of Washington and Martine Schlag of University of California, Santa Cruz for many invaluable discussions and insights. This work was funded in part by NSF under grant number MIP-8806169.