On Reducing Transitions Through Data Modifications

Rajeev Murgai, Masahiro Fujita

Fujitsu Laboratories of America, Inc.
California, USA
murgai,fujita@fla.fujitsu.com

Abstract

Since busses take up significant fraction of chip-area, the bus capacitances are often considerable, and the bus power may account for as much as 40% of the total power consumed on the chip [5]. In applications where the integrity of data is not very important, data may be changed by 3 to 5% without losing too much information. One such application is that of a binary-encoded image, in which case the human eye cannot perceive the small change. However, these small changes can significantly reduce the number of transitions on the data bus and thus the power/energy consumed. We address the following problem: *Given a sequence of n k-bit data words and an error-tolerance e%* (i.e., at most e% of the data bits are permitted to change), *select the bits to be modified so that the total number of transitions is minimized.* We show that a greedy strategy is not always optimum. We propose a linear-time dynamic programming based algorithm that generates an optimum solution to this problem. The experimental results for randomly generated data with a uniform distribution indicate that by changing e% data bits, the transitions can be reduced, on average, by $4e$%.

1 Introduction

With the proliferation of portable electronic devices, low power has become an important design objective. Among the suite of emerging techniques for low power design are pipelining and parallelization [2, 6], reduction of supply voltage [2], and minimization of switching activity. The problem of minimizing the switching activity in a circuit has been addressed at various levels: system level [1, 2], logic or gate level [10, 11, 13], layout level [3, 7, 4], and transistor level [12].

We focus on the issue of minimizing switching activity on high-capacitance busses. In microprocessors and DSPs, busses account for almost 40% of the chip area; 30-40% of the power consumption on a chip is due to bus-power [5], i.e., the power consumed in charging/discharging the often *large* bus capacitances.

Recently, in [8], one aspect of the bus switching activity problem was addressed. If a set of data words or messages is to transmitted over a bus (as shown in Figure 1) such that the sequence in which the words are transmitted is irrelevant, it may be possible to send the data words in an order that minimizes the total number of transitions on the bus. It was shown in [8] that this problem has applications in various design scenarios and phases such as scheduling in high-level synthesis, cache write-back, instruction sequencing, die testing, and transmission of a set of records. In [9], an extra degree of freedom namely word complementation was used to further reduce the transitions.

In this paper, we address a different aspect of switching activity reduction on the bus. In many applications, the integrity of data is not extremely important and the data may be changed by 3 to 5% without incurring a significant information loss. One such application is that of a binary-encoded image (see Figure 2). A typical image may be about 100 x 100 pixels. If each pixel is encoded in (say) 7 bits, the entire image takes 70K bits. Suppose we modify 3% of the binary-encoded image, i.e., change about 2K bits from 0 to 1 (or 1 to 0). Being a relatively small change, it may not be perceived by the human eye. However, the modified bits can significantly reduce the number of transitions seen on the data bus when the image is stored into or read from the memory. Since the data bus is a high-capacitance line, the savings in transitions translate to considerable savings in the power consumed. These savings become significant if the modified data (for instance, the modified image) is stored at a central location (e.g., a popular `ftp` or `www` site). The power savings are available to all the users (possibly hundreds of thousands) who download the data to their respective computers (the bold lines in Figure 2 indicate the sites of power savings). These savings assume even more importance if the computer is a portable one, since the battery life is really critical.

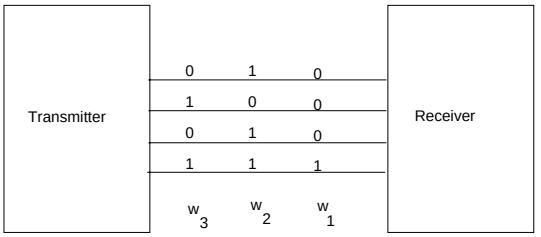

Figure 1: Transmitting a set of words

Example 1.1 *As a simple case, consider the 10-bit sequence S = 0110111100. Assume that S is transmitted on one of the bit-lines of a bus. S has four transitions. Suppose that we are permitted to change (complement or flip, i.e., from 0 to 1 or 1 to 0) 20% of the bits of S, i.e., 2 bits. If we complement the 1st and the 4th bits (both are 0s and we change them to 1s), the new bit-sequence becomes 1111111100, which has just one transition. Thus a 20% modification results in a 75% reduction in the number of transitions.*

The outline of the paper is as follows. In Section 2 we formally state the problem of reducing the number of transitions if a pre-specified number of bits is allowed to flip. We call it the **data modification problem (DMP)**. In Section 3, we present a linear-time algorithm to solve the DMP optimally. Experimental results and implementation details are covered in Section 4. Finally, we conclude with some drawbacks of the proposed methodology and possible fixes in Section 5.

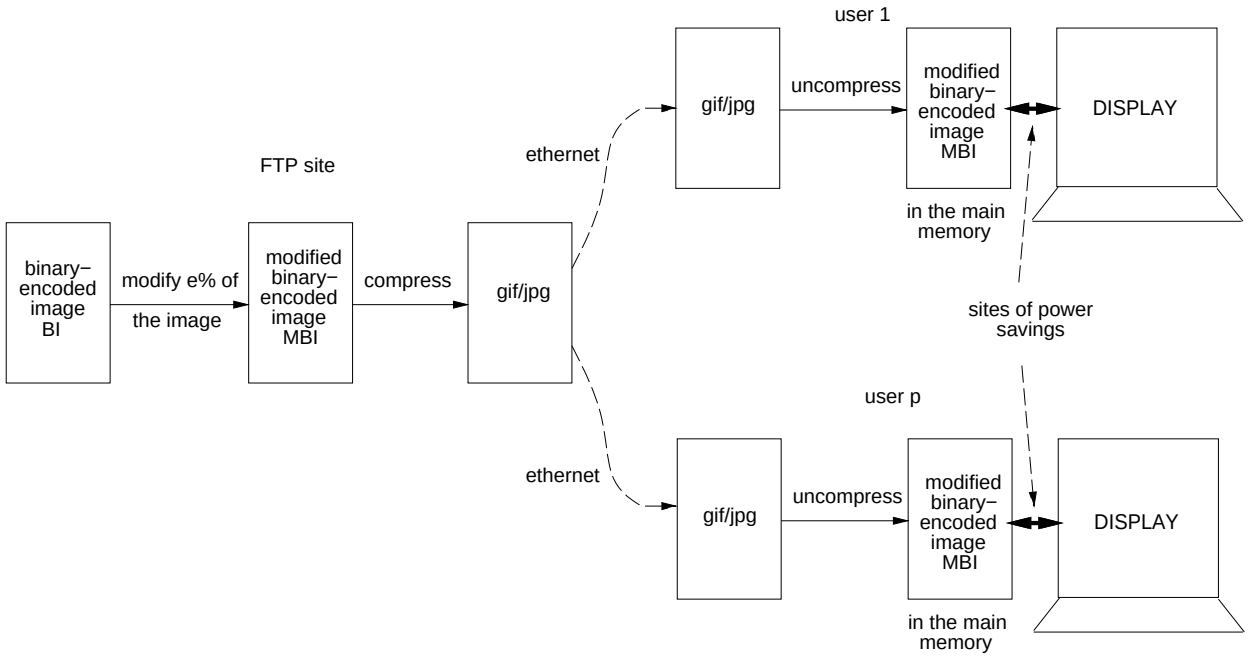

Figure 2: Power savings for a binary-encoded image

2 Problem Definition

Consider n k-bit data words $w_1, w_2, \ldots, w_n$ that have to be transmitted in this order over a k-bit bus, i.e., first w_1 is transmitted, then w_2, and so on, and finally w_n. We are allowed to change at most e bits (out of the total kn bits).[1] The data modification problem (DMP) is deciding which bits to change so that the total number of transitions on the bus is minimized when the modified data words are transmitted in the same order.

If word w_r is transmitted and is immediately followed by w_s, the number of transitions is given by the number of bits that change. This is

$$d(w_r, w_s) = \sum_{j=1}^{k} w_{rj} \oplus w_{sj} \quad (1)$$

sometimes called the **Hamming distance** between w_r and w_s. Here, w_{rj} denotes the j^{th} bit of w_r, and $\oplus$ denotes the Boolean EX-OR.[2] For instance, the Hamming distance between 11001 and 10010 is 3, i.e., $d(11001, 10010) = 3$.

Formally, DMP can be stated as:

"Given a sequence of n k-bit words $w_1, w_2, \ldots w_n$ and an error-tolerance of e bits, find words $\widetilde{w}_1, \widetilde{w}_2, \ldots, \widetilde{w}_n$ such that the total number of transitions, $\sum_{i=1}^{n-1} d(\widetilde{w}_i, \widetilde{w}_{i+1})$, is minimized subject to the constraint that a total of at most e bits change while deriving $\widetilde{w}_i$s from w_is for all i, i.e., $\sum_{i=1}^{n} d(\widetilde{w}_i, w_i) \leq e$."

[1] In the paper, we overload the symbol e to denote both the maximum allowable *percentage error-tolerance* and the maximum *number of bits* that can be modified. The usage should be clear from the context.

[2] Given bits a and b, $a \oplus b = 1$ if $a = 1, b = 0$ or $a = 0, b = 1$.

3 Proposed Solution

First, in Section 3.1, we consider the case of DMP where each word is 1-bit long, i.e., $k = 1$. We call this the **1-bit DMP**. We propose a dynamic programming based algorithm to solve this problem optimally. The algorithm takes time linear in n and the error-tolerance e. Later, in Section 3.2, we address the general problem, i.e., for k-bit data words. We call it the **k-bit DMP**. We extend the algorithm of Section 3.1 to generate an optimum solution for the k-bit DMP in time $O(nke)$.

3.1 The 1-bit DMP

Given n 1-bit words w_i, we think of them as an n-bit long sequence $S = w_1 w_2 \ldots w_n$. We have to choose at most e bits for flipping. There are a total of $\frac{n!}{e!(n-e)!}$ possibilities. For a constant e (i.e., not a function of n), this is $O(n^e)$. However, we do not need to look at all these possibilities. We show that it is possible to devise a simple $O(ne)$ algorithm to solve this problem optimally.

Given an n-bit sequence $S = s_1 s_2 \ldots s_n$, let $\tau(S)$ denote the number of transitions in S, i.e., $\tau(S) = \sum_{i=1}^{n-1} d(s_i, s_{i+1})$.

Definition 3.1 *Given an n-bit sequence S and a positive integer e, an n-bit sequence σ is* **(S, e)-legal** *if $d(S, \sigma) \leq e$ (i.e., σ differs from S in at most e bits).*

Definition 3.2 *Given an n-bit sequence S and a positive integer e, an n-bit sequence $\widetilde{S}$ is* **(S, e)-optimum** *if*

1. *$\widetilde{S}$ is (S, e)-legal, i.e., $d(S, \widetilde{S}) \leq e$, and*
2. *$\tau(\widetilde{S}) = \min\{\tau(\sigma) : \sigma$ an (S, e)-legal sequence$\}$.*

Given a sequence S and error-tolerance e, solving the DMP means finding an (S, e)-optimum sequence. It should be evident that an (S, e)-optimum sequence always exists.

Example 3.1 *Let* S = 110110000, $e = 4$. *Then,* 100000000 *is* $(S,4)$*-legal (in fact, it is* $(S,3)$*-legal), but* 000001100 *is not.* 000000000 *is* $(S,4)$*-optimum.*

We now introduce the concept of a packet.

Definition 3.3 *Given a sequence* S*, a maximal subsequence of* S *containing identical bits is called a* **packet**. *A* **0-packet** *has all bits 0, and a* **1-packet** *has all 1s. We will represent a 0-packet by* $\vec{0}$ *and a 1-packet by* $\vec{1}$. *Let* $|p|$ *denote the number of bits in the packet* p.

A completely-flipped packet $\overline{p}$ *is obtained by flipping (or complementing) all the bits of the packet* p. *Given a packet* p *with at least 2 bits, a* **partially-flipped packet** *is obtained from* p *by flipping at least one but not all bits.*

Note that after partial flipping, a packet has at least one 0-bit and at least one 1-bit.

Example 3.2 *Let* S = 110110000. *Then,* S *has 4 packets (from left to right):* $p_1 = 11, p_2 = 0, p_3 = 11$ *and* $p_4 = 0000$. *Thus,* S *has a unique* **packet decomposition***:* $p_1p_2p_3p_4$. $\overline{p_4} = 1111$. *An example of a partially-flipped packet obtained from* p_4 *is 0010.*

The concept of a packet is an important one, since as we show next, there exists an optimum sequence in which no packet is partially flipped, i.e., either a packet remains intact or is completely flipped.

Given a packet p of S, let p^ℓ denote the leftmost bit of p and p^r the rightmost bit of p. Let b^ℓ denote the bit to the left of p^ℓ and b^r the bit to the right of p^r. So S looks like

$$S = \ldots b^\ell \underbrace{p^\ell \ldots p^r}_{p} b^r \ldots$$

Note that b^ℓ or b^r may be NULL if p is the first or the last packet respectively.

Proposition 3.1 *Given a sequence* S *with its packet decomposition* $p_1p_2\ldots p_z$, *and* e, *the maximum number of modifiable bits. Then, there exists an* (S,e)*-optimum sequence in which no packet of* S *is partially flipped.*

Proof Let $V = \{\sigma : \sigma$ is an (S,e)-optimum sequence with at least one packet of S partially flipped $\}$. $V = \{\}$ implies the proposition. So assume V is non-empty. Then, there exists $S_1 \in V$ such that S_1 is an (S,e)-optimum sequence with the minimum number of partially flipped packets. We will derive an (S,e)-optimum sequence $\widetilde{S}$ from S_1 that does not have any partially-flipped packets. Let p be a partially-flipped packet in S_1. By definition, $|p| \geq 2$. We can safely assume $|S| > |p|$ (i.e., S has at least two packets), otherwise p is the only packet of S and flipping some but not all bits is clearly non-optimum. Without loss of generality, assume p is a 0-packet. Then, b^ℓ and b^r, if non-NULL, are each 1 (from the maximality of the packet p).

Depending upon p's place in S, there are three cases:

1. p is the first (i.e., leftmost) packet of S: There are four subcases depending on the values of p^r and b^r.

   (a) In S_1, $p^r = 1$, and $b^r = 1$: Because p is partially flipped, it has a 0 bit somewhere in S_1. S_1 then looks like

   $$S_1 = \underbrace{\ldots 0 \ldots \overbrace{1}^{p^r}}_{p} \underbrace{1}_{b^r} R \quad \Big)_{pb^r}$$

   R denotes the remaining sub-sequence of S_1. Then in S_1, $\tau(pb^r) \geq 1$. Consider the following sequence $\widetilde{S}$, which is the same as S_1 except that no bit of p is flipped (i.e., all of p's bits are 0):

   $$\widetilde{S} = \underbrace{\vec{0}}_{p} \underbrace{1}_{b^r} R$$

   Clearly, in $\widetilde{S}$, $\tau(pb^r) = 1 \leq \tau(pb^r)$ in S_1. Since the subsequences b^rR are identical in $\widetilde{S}$ and S_1, $\tau(\widetilde{S}) \leq \tau(S_1)$. Since the number of bit-flips in $\widetilde{S}$ is less than that in S_1, (S,e)-legality of S_1 implies that $\widetilde{S}$ is (S,e)-legal.

   (b) In S_1, $p^r = 0$, and $b^r = 1$: Since p is partially-flipped, it has a 1 bit somewhere in S_1. S_1 then looks like

   $$S_1 = \underbrace{\ldots 1 \ldots \overbrace{0}^{p^r}}_{p} \underbrace{1}_{b^r} R$$

   Then, in S_1, $\tau(pb^r) \geq 2$. Consider the following sequence $\widetilde{S}$, which is the same as S_1, except that no bit of p is flipped:

   $$\widetilde{S} = \underbrace{\vec{0}}_{p} \underbrace{1}_{b^r} R$$

   In $\widetilde{S}$, $\tau(pb^r) = 1$. Thus, $\tau(\widetilde{S}) < \tau(S_1)$. As before, $\widetilde{S}$ is (S,e)-legal.

   (c) In S_1, $p^r = 1$, and $b^r = 0$: can be handled similarly.

   (d) In S_1, $p^r = 0$, and $b^r = 0$: can be handled similarly.

2. p is the last packet of S: this is symmetric to the previous case.

3. p is somewhere in the middle of S: There are four cases depending on the values of b^ℓ and b^r. We will only address the first case; the rest are proved identically.

   (a) In S_1, both $b^\ell = b^r = 1$. S_1 can be written as either

   $$S_1 = L \overbrace{1}^{b^\ell} \underbrace{\ldots 0 \ldots 1 \ldots}_{p} \overbrace{1}^{b^r} R$$

   or

   $$S_1 = L \overbrace{1}^{b^\ell} \underbrace{\ldots 1 \ldots 0 \ldots}_{p} \overbrace{1}^{b^r} R$$

   L and R are the subsequences in S_1 to the left of b^ℓ and to the right of b^r respectively. Consider a sequence $\widetilde{S}$ which is identical to S_1, except that no bit of p is flipped.

   $$\widetilde{S} = L \overbrace{1}^{b^\ell} \underbrace{\vec{0}}_{p} \overbrace{1}^{b^r} R$$

   Then, in $\widetilde{S}$, $\tau(b^\ell pb^r) = 2$. Since in S_1 at least one 0 bit and at least one 1 bit occur in the packet p, in S_1, $\tau(b^\ell pb^r) \geq 2$. Thus, $\tau(\widetilde{S}) \leq \tau(S_1)$. As before, $\widetilde{S}$ is (S,e)-legal.

   (b) In S_1, both $b^\ell = b^r = 0$.

(c) In S_1, $b^\ell = 0, b^r = 1$.

(d) In S_1, $b^\ell = 1, b^r = 0$.

In each case, we derived a sequence $\widetilde{S}$ such that

1. $\widetilde{S}$ does not have p partially flipped. In fact, p remains what it was in S, and
2. except p, all packets are identical in S_1 and $\widetilde{S}$, and
3. $\widetilde{S}$ is (S,e)-legal, since S_1 is (S,e)-legal and S_1 flips some bit(s) of p whereas $\widetilde{S}$ does not (the other parts of S_1 and $\widetilde{S}$ are identical), and
4. $\tau(\widetilde{S}) \leq \tau(S_1)$.

Then it follows that since S_1 is (S,e)-optimum, $\widetilde{S}$ is also (S,e)-optimum. However, $\widetilde{S}$ has one less packet partially flipped as compared to S_1. From the minimality of S_1 with respect to the number of partially-flipped packets, it follows that $\widetilde{S} \notin V$. That implies $\widetilde{S}$ has no partially-flipped packet. Thus, we have derived an (S,e)-optimum sequence $\widetilde{S}$ that does not have any partially-flipped packet. ■

In other words, Proposition 3.1 says that there exists an optimum sequence in which each original packet is either unchanged or completely flipped. So we can restrict the problem of selecting from S at most e bits for flipping to that of selecting packets for complete flipping, and still be able to obtain an optimum solution. The selected packets should not flip more than a total of e bits. To formalize all this, we introduce the notions of a packet-selection set, its legality, and optimality.

Definition 3.4 *Given an n-bit sequence S with packet decomposition $S = p_1 p_2 \ldots p_z$. A* **packet-selection set** *U is a subset of $\{p_1, p_2, \ldots p_z\}$ and implies that all the packets in U will be completely flipped, and a packet not in U will remain completely unchanged. Let $U(S)$ denote the sequence obtained from S by completely flipping all the packets of U.*

Note that $d(S, U(S)) = \sum_{p \in U} |p|$.

Definition 3.5 *Given an n-bit sequence S with packet decomposition $S = p_1 p_2 \ldots p_z$, and a positive integer e. A packet-selection set U is* **(S,e)-legal** *if $U(S)$ is (S,e)-legal, i.e., $(\sum_{p \in U} |p|) \leq e$.*

A packet-selection set U is **(S,e)-optimum** *if $U(S)$ is (S,e)-optimum.*

Example 3.3 *Consider the sequence $S = 110110000$. The packet decomposition of S is $p_1 = 11, p_2 = 0, p_3 = 11, p_4 = 0000$. Let $U = \{p_1, p_2\}$. Then, $U(S) = 001110000$. Note that U is $(S,3)$-legal since p_1 and p_2 have a total of 3 bits. Clearly, U is not $(S,2)$-legal. Note that $U = \{p_2\}$ is $(S,1)$-, $(S,2)$-, and $(S,3)$-optimum. $\{p_3\}$ and $\{p_2, p_3\}$ are also $(S,3)$-optimum.*

Our goal now is to determine an (S,e)-optimum packet-selection set U. Before that, we introduce the notions of packet cost and packet gain.

Definition 3.6 *With each packet p, we associate a* **packet cost** *$C(p)$, which is the number of bits in the packet (i.e., $|p|$), and a* **packet gain** *$G(p)$, which is the reduction in the number of transitions when p is completely flipped and the packets adjacent to p remain the same.*

Note that $G(p) = 0$ if p is the only packet in S, $G(p) = 1$ if p is the first or the last packet of S, otherwise $G(p) = 2$.

Next we show that a simple greedy strategy is not always optimum.

3.1.1 Greedy Algorithm is Sub-Optimum

A greedy algorithm proceeds in stages. Let v denote the number of bits that can be flipped at a given stage in the algorithm. Initially, $v = e$. At each stage, the algorithm selects *the best packet p* for flipping. The best packet is the one that has the maximum non-zero gain among all the packets that have a cost of at most v and if there is more than one such packet, p is the one with the minimum cost (i.e., with the minimum number of bits). After flipping p, at most $v - C(p)$ more bits can be flipped. The algorithm stops when it cannot find a v-legal packet with non-zero gain.

As the following example illustrates, the greedy algorithm may not always yield the optimum solution.

Example 3.4 *Consider the sequence $S = 110110000$, with $e = 4$. Let us divide the sequence into its packets, along with their gains and costs:*

packet p_i	*gain $G(p_i)$*	*cost $C(p_i) = \|p_i\|$*
$p_1 = 11$	1	2
$p_2 = 0$	2	1
$p_3 = 11$	2	2
$p_4 = 0000$	1	4

The greedy algorithm would work as follows. Initially, $v = e = 4$. Each packet is $(S,4)$-legal. Packets p_2 and p_3 have the maximum gain. Since $C(p_2) < C(p_3)$, p_2 is picked for flipping. The new sequence is $S_1 = 111110000$. Now, $v = 4 - C(p_2) = 3$. S_1 has two packets 11111 and 0000, but neither is $(S_1,3)$-legal. Hence, no more transition reduction is possible. $\tau(S_1) = 1$.

However, an $(S,4)$-optimum sequence $\widetilde{S}$ is obtained by $U = \{p_1, p_3\}$. $\widetilde{S} = U(S) = 000000000$; $\tau(\widetilde{S}) = 0$. Note that U is $(S,4)$-legal.

3.1.2 An Optimum Dynamic Programming Algorithm

We first prove that there exists an optimum packet-selection set that does not contain any two adjacent packets of S.

Proposition 3.2 *Given a sequence S with its packet decomposition $p_1 p_2 \ldots p_z, z \geq 2$ and e, the maximum number of modifiable bits. Then, there exists an (S,e)-optimum packet-selection set $\widetilde{U}$ such that for no i, $1 \leq i < z$, both $p_i, p_{i+1} \in \widetilde{U}$.*

Proof Let $V = \{W : W$ is an (S,e)-optimum packet-selection set containing at least one adjacent pair of packets of $S\}$.

If $V = \{\}$, we are done. Otherwise, let $U \in V$ contain the minimum number of adjacent packet pairs as compared to all the sets in V. Let $p_i, p_{i+1} \in U$ for some i. Without loss of generality, assume that in S, p_i is a 0-packet (denoted $\vec{0}$). Then p_{i+1} is a 1-packet (denoted $\vec{1}$), p_{i-1} a 1-packet (if it exists) and p_{i+2} a 0-packet (if it exists).

There are four cases:

1. Both p_{i-1} and p_{i+2} are NULL, i.e., $i = 1$ and S has only two packets. $S = p_i p_{i+1} = \vec{0}\vec{1}$. Then $U = \{p_i, p_{i+1}\}$ and $U(S) = \overline{p_i p_{i+1}} = \vec{1}\vec{0}$. Then, $\tau(U(S)) = 1$ and $d(U(S), S) = |p_i| + |p_{i+1}|$. Consider the set $\widetilde{U} = U - \{p_{i+1}\} = \{p_i\}$. Then, $\widetilde{U}(S) = \overline{p_i} p_{i+1} = \vec{1}\vec{1}$. So $\tau(\widetilde{U}(S)) = 0 < \tau(U(S))$. Also, $d(\widetilde{U}(S), S) = |p_i| < |p_i| + |p_{i+1}| = d(U(S), S)$.
2. p_{i-1} is NULL, but p_{i+2} is not, i.e., $i = 1$ and S has at least three packets, i.e.,

$$S = \underbrace{\vec{0}}_{p_i} \underbrace{\vec{1}}_{p_{i+1}} \underbrace{\vec{0}}_{p_{i+2}} \ldots$$

There are two sub-cases:

(a) $p_{i+2} \notin U$: Then

$$U(S) = \underbrace{\vec{1}}_{p_i}\underbrace{\vec{0}}_{p_{i+1}}\underbrace{\vec{0}}_{p_{i+2}}R$$

Here R is the sub-sequence of $U(S)$ to the right of p_{i+2}. Let $\widetilde{U} = U - \{p_i\}$. Then

$$\widetilde{U}(S) = \underbrace{\vec{0}}_{p_i}\underbrace{\vec{0}}_{p_{i+1}}\underbrace{\vec{0}}_{p_{i+2}}R$$

Clearly, $\tau(\widetilde{U}(S)) = \tau(U(S)) - 1 < \tau(U(S))$. Also, $\widetilde{U}$ is (S,e)-legal, since U is (S,e)-legal and $\widetilde{U} \subset U$.

(b) $p_{i+2} \in U$: Then

$$U(S) = \underbrace{\vec{1}}_{p_i}\underbrace{\vec{0}}_{p_{i+1}}\underbrace{\vec{1}}_{p_{i+2}}R$$

Let $\widetilde{U} = U - \{p_{i+1}\}$. Then,

$$\widetilde{U}(S) = \underbrace{\vec{1}}_{p_i}\underbrace{\vec{1}}_{p_{i+1}}\underbrace{\vec{1}}_{p_{i+2}}R$$

$\tau(\widetilde{U}(S)) = \tau(U(S)) - 2 < \tau(U(S))$. Also, $\widetilde{U}(S)$ is (S,e)-legal.

3. p_{i+2} is NULL, but p_{i-1} is not: this is symmetric to the last case.

4. Both p_{i-1} and p_{i+2} are non-NULL: there are four sub-cases here:

(a) $p_{i-1}, p_{i+2} \in U$: Then

$$U(S) = L\underbrace{\vec{0}}_{p_{i-1}}\underbrace{\vec{1}}_{p_i}\underbrace{\vec{0}}_{p_{i+1}}\underbrace{\vec{1}}_{p_{i+2}}R$$

Let $\widetilde{U} = U - \{p_i\}$:

$$\widetilde{U}(S) = L\underbrace{\vec{0}}_{p_{i-1}}\underbrace{\vec{0}}_{p_i}\underbrace{\vec{0}}_{p_{i+1}}\underbrace{\vec{1}}_{p_{i+2}}R$$

(b) $p_{i-1} \in U, p_{i+2} \notin U$: can be handled similarly.

(c) $p_{i-1} \notin U, p_{i+2} \in U$: can be handled similarly.

(d) $p_{i-1}, p_{i+2} \notin U$: can be handled similarly.

In each of the above cases, we derived a packet-selection set $\widetilde{U}$ such that

1. $\widetilde{U} = U - \{p_j\}, j \in \{i, i+1\}$. So $\widetilde{U}$ does not contain the packet-pair $\{p_i, p_{i+1}\}$ and $\widetilde{U}$ is (S,e)-legal, and
2. $\tau(\widetilde{U}(S)) \leq \tau(U(S))$.

Then, from the (S,e)-optimality of U, it follows that $\widetilde{U}$ is also (S,e)-optimum, with at least one less pair of adjacent packets than U. The minimality of U in V with respect to the adjacent pairs implies that $\widetilde{U} \notin V$. In other words, $\widetilde{U}$ does not have any pair of adjacent packets. Hence proved. ■

Now, we are ready to present the optimum algorithm for the 1-bit DMP, which selects a set of packets for complete flipping such that the total cost of selected packets is at most e and the reduction in the number of transitions is maximized. The resulting sequence would have the minimum number of transitions.

Let $S = p_1 p_2 \ldots p_z$. Let $TG(p_i, v)$, $v \leq e$, denote the maximum transition gain (or reduction) when only a subset of packets from $\{p_i, p_{i+1}, \ldots, p_z\}$ is selected with a total of at most v bit-flips permitted.

$$TG(p_i, v) = \begin{cases} 0 & \text{if } v \leq 0 \\ TG(p_{i+1}, v) & \text{if } C(p_i) > v \\ \max\{G(p_i) + TG(p_{i+2}, v - C(p_i)), & \\ \quad TG(p_{i+1}, v)\} & \text{otherwise} \end{cases} \quad (2)$$

We set $TG(p_{z+1}, v) = TG(p_{z+2}, v) = 0 \ \forall \ v$.

(2) may be understood as follows. If the cost of packet p_i is greater than the maximum number of bits, v, allowed to be modified, p_i cannot be selected in the solution, and the total transition gain is equal to the total transition gain from the packet p_{i+1} onwards with a budget of v modifiable bits. Otherwise, there are two possibilities:

- p_i *is selected*: the total gain is the gain in transitions due to p_i (i.e., $G(p_i)$) plus the total gain from the packet p_{i+2} onwards with a budget of $v - C(p_i)$ modifiable bits. Note that since p_i has already been selected, p_{i+1} is not selected – from Proposition 3.2. Recall that if p_i is selected, the definition of the packet gain $G(p_i)$ assumes that the packets adjacent to p_i (i.e., p_{i+1} and p_{i-1}) are not selected. We just saw that p_{i+1} is not selected. We note that if p_{i-1} is selected, p_i is not selected. In other words, if p_i is selected, p_{i-1} is not selected. So we are correct in adding the gain $G(p_i)$ to $TG(p_{i+2}, v - C(p_i))$ while computing $TG(p_i, v)$.
- p_i *is not selected*: the gain is the total accumulated gain from p_{i+1} onwards, with the total budget still at v.

The gain $TG(p_i, v)$ is the maximum of these two cases.

The optimum solution to the 1-bit DMP then corresponds to the value of $TG(p_1, e)$. If at each packet p_i, along with $TG(p_i, v)$, we also remember whether the best solution selects p_i, we can determine the optimum packet-selection set.

Computation Complexity: For each packet p_i, we need to store the TG values for $1 \leq v \leq e$ in the worst case.[3] Note from (2) that the computation of $TG(p_i, v)$ takes constant time provided $TG(p_{i+1}, v)$ and $TG(p_{i+2}, v - C(p_i))$ are already computed. Also, once a TG value is computed for a packet p_i and budget v, it need not be computed again. Since there are z packets, the time complexity of the algorithm is given by $O(ze)$. Since $z \leq n$, the complexity is $O(ne)$.

The above discussion leads us to the following proposition.

Proposition 3.3 *Given n 1-bit data words and that at most e bits can change, the algorithm described above computes the optimum solution to the 1-bit DMP.*

Example 3.5 *Let $S = 110110000$, $e = 4$. The packets p_i, their costs $C(p_i)$ and gains $G(p_i)$ were shown in Example 3.4.*

$TG(p_1, 4) = \max\{1 + TG(p_3, 2), TG(p_2, 4)\}$

- $TG(p_3, 2) = \max\{2 + TG(p_5, 0), TG(p_4, 2)\}$
  - $TG(p_5, 0) = 0$.
  - $TG(p_4, 2) = 0$ *(because $|p_4| = 4$).*

  Thus, $TG(p_3, 2) = \max\{2 + 0, 0\} = 2$ – *select* p_3.
- $TG(p_2, 4) = \max\{2 + TG(p_4, 3), TG(p_3, 4)\}$
  - $TG(p_4, 3) = 0$ *(because $|p_4| = 4$).*
  - $TG(p_3, 4) = \max\{2 + TG(p_5, 2), TG(p_4, 4)\} = \max\{2 + 0, 1\} = 2$ – *select* p_3.

  Thus, $TG(p_2, 4) = \max\{2 + 0, 2\} = 2$.

Thus, $TG(p_1, 4) = \max\{1 + 2, 2\} = 3$ – *select* p_1.

In the optimum solution, we select p_1 and p_3, or $\widetilde{U} = \{p_1, p_3\}$. The total reduction in transitions is $TG(p_1, 4) = 3$. The resulting optimum sequence $\widetilde{U}(S) = 000000000$, as noted in Example 3.4.

[3] In practice, we observed that much fewer values are needed.

3.2 The General Case: k-bit DMP

We now remove the restriction that each data word be a single bit. Each word is k-bit wide, $k \geq 1$. It turns out that the dynamic programming algorithm for $k = 1$ can be easily extended for solving the k-bit DMP optimally as follows.

We are given n k-bit words: $w_1, w_2, \ldots w_n$, where

$$w_i = w_{i1} w_{i2} \ldots w_{ik} \tag{3}$$

For each bit $j, 1 \leq j \leq k$, form the sequence S_j by concatenating the j^{th} bits of all the words as follows:

$$S_j = w_{1j} w_{2j} \ldots w_{nj} \tag{4}$$

For each S_j, determine its packet decomposition and store the cost and gain of each packet. Finally, concatenate all S_js by inserting a dummy packet $\pi_{j,j+1}$ between the sequences S_j and S_{j+1}. Assign a cost of 0 and a gain of 0 to each dummy packet. Let the resulting sequence be S.

$$S = S_1 \ (\pi_{1,2}) \ S_2 \ (\pi_{2,3}) \ S_3 \ldots S_{k-1} \ (\pi_{k-1,k}) \ S_k \tag{5}$$

Note that the packet decomposition of S is given by the packet decomposition of S_1 followed by $\pi_{1,2}$ followed by the packet decomposition of S_2, and so on. Apply the algorithm of Section 3.1.2 on this packet decomposition of S and obtain the (S, e)-optimum packet-selection set U. From U, discard all the dummy packets π. The resulting set of packets is the optimum solution to the original k-bit DMP. It can be seen that the complexity of the algorithm is $O(nke)$.

Proposition 3.4 *The algorithm presented above computes the optimum solution to the k-bit DMP.*

Sketch of Proof We need to ensure that the selection of S_j's last packet does not exclude the selection of S_{j+1}'s first packet (the exclusion would have occurred had we simply concatenated the packet decompositions of the individual sequences – see (2)). The presence of the dummy packet $\pi_{j,j+1}$ accomplishes exactly that. Also, if the algorithm selects some dummy packet $\pi_{j,j+1}$, the cost and transition gain of the final solution (after throwing away $\pi_{j,j+1}$) are not changed, since $\pi_{j,j+1}$ has zero cost and zero gain. ■

n	τ	e (% of n)	e (bits)	$\Delta\tau$	$\Delta\tau$ (%)
10	7	10	1	2	28
10	4	10	1	2	50
10	3	10	1	2	66
40	20	3	1	2	10
40	23	3	1	2	8
40	17	3	1	2	11
40	18	10	4	8	44
40	20	10	4	8	40
40	16	10	4	8	50
100	50	3	3	6	12
100	45	3	3	6	13
100	55	3	3	6	10
100	58	10	10	20	34
100	44	10	10	20	45
100	48	10	10	20	41
500	246	3	15	30	12
500	290	3	15	30	10
500	247	3	15	30	12
500	261	10	50	100	38
500	276	10	50	100	36
500	267	10	50	100	37
1000	508	3	30	60	11
1000	491	3	30	60	12
1000	525	3	30	60	11
1000	487	10	100	200	41
1000	528	10	100	200	37
1000	506	10	100	200	39

n the number of 1-bit words
τ the total number of transitions in the n-bit sequence
e the error-tolerance
$\Delta\tau$ reduction in # transitions after data modification

Table 1: Reduction in Transitions

4 Experimental Results

The objective of our experiments is to evaluate the effectiveness of the proposed dynamic programming based algorithm in minimizing the switching activity. The experimental set-up is as follows. Since the general k-bit problem is essentially the same as the 1-bit problem, we only consider n 1-bit data words. Further, these words are generated from a uniform distribution: the probabilities of generating a 1 and a 0 are the same (= 0.5). We count the total number of transitions (column τ in Table 1) if the words are transmitted without modification and in the order generated. Next, we specify the error-percentage e, and from it compute the maximum number of bits allowed to flip, denoted as *e (bits)*. We apply the algorithm of Section 3.1.2 on the n-bit sequence and determine the reduction in transitions (columns $\Delta\tau$ and $\Delta\tau$ (%)). The experiment is run for different values of n and e. For each (n, e) pair, 3 different bit sequences are generated.

The results are shown in Table 1. The number of bits (or words), n, is varied as 10, 40, 100, 500, and 1000. The maximum number of bits permitted to change, e, is set to 3 and 10% of n. Comparing the columns "e (% of n)" and $\Delta\tau$ (%), it can be deduced that *if e% bits are allowed to flip, the average percentage reduction in transitions is about* $4e$. This is indeed encouraging, since a small distortion in the data can reduce the transitions significantly. We can explain this behavior analytically as follows. Let the probability of occurrence of a 1 in the bit stream be p. Then the probability of a 0 is $(1 - p)$. There is a transition from the bit i to bit $i + 1$ if the bit i is 1 and $i + 1$ is 0, or the bit i is 0 and $i + 1$ is 1. The corresponding transition probability is then $p(1 - p) + (1 - p)p = 2p(1 - p)$. The expected number of transitions in an n bit sequence $E(\tau) \sim 2np(1 - p)$. Now, e% bits of $n = en/100$ bits are allowed to change. Given that e is small, the algorithm can usually find packets each with a cost of 1 and a transition gain of 2. Thus, the savings in transitions $\Delta\tau$ are about $2en/100$. The percentage reduction in transitions is then

$$\Delta\tau(\%) = \frac{2en/100}{2np(1-p)} 100 \tag{6}$$

$$= \frac{e}{p(1-p)} \tag{7}$$

Setting $p = 0.5$ (as is the case in the bit sequences generated for the above experiment), we obtain a reduction of about $4e$. To further confirm the analysis, we conducted the entire experiment with $p = 0.25$. We do not report the results here, but we observed a percentage reduction of about $5e$, which is consistent with $16e/3$

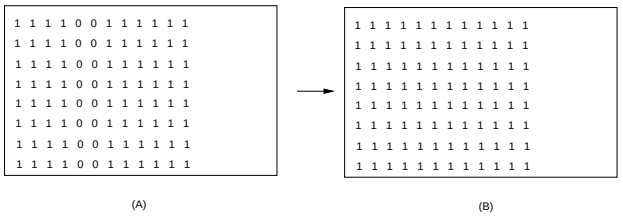

Figure 3: A pathological case when our algorithm can destroy useful information

given by (7).[4]

5 Discussion

In this paper, we identified an avenue for reducing switching activity in applications where data integrity is not crucial. One application where this may be useful is in storing and displaying images on the computer. We presented an optimum linear-time dynamic programming based algorithm to solve this problem of data modification for minimizing the total number of transitions given an upper bound on the number of modifiable bits. Experimental results on randomly generated data with uniform distribution show that by modifying data by $e\%$, on average one can hope to reduce transitions by $4e\%$.

Our technique has at least one drawback. Consider a binary-encoded black and white picture, shown in Figure 3 (A). A '1' represents a white pixel, and '0' a black one. Thus a black strip runs down the entire length of the original picture. Suppose each row is transferred on a data line. If 20% distortion is allowed, our algorithm will change each 0 into a 1, reducing the number of transitions by 100%. However, as a result, the black strip disappears. The entire picture is now white (i.e., all 1s), as shown in Figure 3 (B)! The problem arises because the algorithm can modify *any* data bits in order to maximize the transition reduction. It does not distinguish portions of the data stream that have high information content (e.g., object boundaries) from those having low information content (e.g., the continuous white background). The problem may be rectified by marking the high-information regions and disallowing the algorithm to modify them. We plan to work on this extension in the future. We must point out, however, that such an extreme behavior should manifest itself rarely, the reason being that typically each pixel is encoded using 7 bits, which represent the color-coding or the gray-level of the image at that location. So a 1 to 0 (or 0 to 1) transition in the data stream may not correspond to a sharp boundary in the image (since a boundary is an *inter-pixel* manifestation), and may just be within the pixel (i.e., *intra-pixel*). When the algorithm changes the subsequence 1001 to 1111, it may not necessarily be making changes across two pixels, but simply within a pixel, in which case, the boundary is not destroyed.

In future, we plan to apply our algorithm on real binary-coded images and audio streams and study the reduction in transitions vis-a-vis degradation in the quality.

[4] Note that (7) should be applied only when the expected savings in transitions $\Delta\tau = 2en/100$ is not more than the total number of transitions $\tau = 2np(1-p)$. For instance, when $p = 0$ or $p = 1$, the total number of transitions (the denominator in (6)) is 0, and hence there can be no savings in transitions. However, (7) predicts huge savings!

Finally, we realize that data compression is an obvious approach to saving power. Our data modification technique is complementary to data compression, in that it can be applied in tandem with compression (as was shown in Figure 2), and thus save additional power.

References

[1] J. Bunda, W. C. Athas, and D. Fussell. Evaluating Power Implications of CMOS Microprocessor Design Decisions. In *International Workshop on Low Power Design*, pages 147–152, April 1994.

[2] A. Chandrakasan, T. Sheng, and R. W. Brodersen. Low Power CMOS Digital Design. In *Journal of Solid State Circuits*, pages 473–484, April 1992.

[3] K. Y. Chao and D. F. Wong. Low Power Considerations in Floorplan Design. In *International Workshop on Low Power Design*, pages 45–50, April 1994.

[4] J. Cong, C. K. Koh, and K. S. Leung. Wiresizing with Driver Sizing for Performance and Power Optimization. In *International Workshop on Low Power Design*, pages 81–86, April 1994.

[5] Experienced Motorola Designer. Personal comm., April 1995.

[6] P. Duncan, S. Swamy, and R. Jain. Low-Power DSP Circuit Design Using Retimed Maximally Parallel Architectures. In *Proceedings of the* 1^{st} *Symposium on Integrated Systems*, pages 266–275, March 1993.

[7] V. Hirendu and M. Pedram. PCUBE: A Performance Driven Placement Algorithm for Lower Power Designs. In *Proceedings of Euro-DAC*, pages 72–77, 1993.

[8] R. Murgai, M. Fujita, and S. C. Krishnan. Data Sequencing for Minimum-transition Transmission. In *VLSI'97*, August 1997.

[9] R. Murgai, M. Fujita, and A. Oliveira. Using Complementation and Resequencing To Minimize Transitions. In *Proceedings of the Design Automation Conference*, pages 694–697, June 1998.

[10] K. Roy and S. Prasad. SYCLOP: Synthesis of CMOS Logic for Low Power Applications. In *Proceedings of the Int'l Conference on Computer Design: VLSI in Computers and Procesors*, pages 464–467, October 1992.

[11] A. Shen, S. Devadas, J. White, A. Ghosh, and K. Keutzer. A Combinational Logic Design Methodology Targeting Low Power Applications. In *MIT Technical Report (available from the authors)*, February 1993.

[12] C. H. Tan and J. Allen. Minimization of Power in VLSI Circuits Using Transistor Sizing, Input Ordering, and Statistical Power Estimation. In *International Workshop on Low Power Design*, pages 75–80, April 1994.

[13] V. Tiwari, P. Ashar, and S. Malik. Technology Mapping for Low Power. In *Proceedings of the* 30^{th} *Design Automation Conference*, pages 74–79, June 1993.