Power-optimal Encoding for a DRAM Address Bus

Wei-Chung Cheng and Massoud Pedram

***Abstract*-- This paper presents an irredundant encoding technique to minimize the switching activity on a multiplexed Dynamic RAM (DRAM) address bus. The DRAM switching activity can be classified either as external (between two consecutive addresses) or internal (between the row and column addresses of the same address). For external switching activity in a sequential access pattern, we present a power-optimal encoding, named Pyramid code. Extensions of the basic code address different types of DRAM devices. The proposed codes reduce power dissipation on the memory bus by a factor of two or more.**

***Index Terms*— address bus encoding, DRAM power minimization, time-multiplexed bus, bus activity minimization**

I. INTRODUCTION

Modern electronic systems must maintain a challenging dichotomy; they need to be low power and high performance simultaneously. This arises largely from their use in battery-operated portable (wearable) platforms. Even in fixed, power-rich platforms, the packaging and reliability costs associated with very high power and high performance systems are forcing designers to look for ways to reduce power consumption. Power-efficient designing requires reducing power dissipation in all parts of the design and during all stages of the design process subject to constraints on system performance and quality of service (QoS). Sophisticated power-aware, high-level language compilers, dynamic power management policies, memory management and bus encoding techniques, as well as hardware design tools are demanded to meet these often conflicting design requirements [1] [2]. This paper focuses on the low power bus-encoding problem. In this section, we will briefly review the bus encoding techniques and DRAM device technology.

The major building blocks of a computer system include the CPU, the memory controller, the memory chips, and the communication channels dedicated to providing the means for data transfer between the CPU and the memory. These channels tend to support heavy traffic and often constitute the performance bottleneck in many systems. At the same time, the energy dissipation per memory bus access is quite high, which in turn limits the power efficiency of the overall system.

In a computer system, the bus can be an on-chip bus, a local bus between the CPU and the memory controller, or a memory bus between the memory controller (which may be on-chip or off-chip) and the memory devices. The emphasis of this paper is on low power encoding techniques for the memory bus. We assume the availability of a separate code memory and data memory. More precisely, we present encoding techniques to minimize the switching activity on a multiplexed DRAM address bus.

In the remainder of this section, we give a detailed review of the power-aware bus encoding techniques followed by a summary description of the DRAM technology. We provide the problem formulation for external switching activity minimization in conventional DRAM in Section II. Pyramid II code, which uses a much simpler encoding function, is presented in Section III. Extensions to handle Burst mode DRAM are described in Section IV. Concluding remarks are provided in Section V.

A. Memory Bus Encoding

Low power bus codes can be classified as *permutation*, *algebraic*, or *probabilistic*. Permutation codes refer to a permutation of a set of source words. Algebraic codes refer to codes that are produced by encoders that take two or more operands (e.g., the current source word, the previous source word, the previous code word, etc.) to produce the current code word using arithmetic or bit-level logic operations. Probabilistic codes are generated by encoders that examine the probability distribution of source words or pairs of source words and use this distribution to assign codes to the source words or pairs of source words. In all cases, the objective is to minimize the number of transitions when transmitting all of the code words on the bus. The overhead of the encoder/decoder circuitry is often ignored.

If redundancy is not feasible on the memory bus, the encoding function becomes a permutation, i.e., a one-to-one and on-to mapping from a set of source words to itself. In [3], Su, Tsui, and Despain proposed Gray code to implement the program counter of a microprocessor to minimize the switching activity of sequential memory

W. C. Cheng and M. Pedram are with the Department of Electrical Engineering–Systems, University of Southern California, Los Angeles, CA 90089-2560 USA.

accesses. They showed that Gray code is asymptotically optimal among all irredundant codes. Other examples include Pyramid code [4][5] and Data Ordering-based code [6][7].

Bus-Invert code [8] toggles the polarity of the signals according to the Hamming distance between two consecutive data values by using an additional line on the bus. Many variations of Bus-Invert code have been proposed in the literature, including Partial Bus-Invert code [9], Interleaving Partial Bus-Invert code [10], and Two-dimensional code [11]. Similarly, T0 code [12] uses a redundant signal to indicate if the bus is in normal mode or increasing address. In the latter case, only one signal needs to be switched. Variations on the T0 code include T0-XOR and Offset-XOR codes [13]. Both Bus-Invert and T0 codes are redundant because they need one extra bit. T0 code is not suitable for reducing bus activity in a time-multiplexed address bus. A *k*-limited-weight code [14] is a code having at most *k* one's per word. This can be achieved by adding appropriate redundant lines. These codes are useful in conjunction with transition signaling. Thus, a *k*-limited-weight code would guarantee at most *k* transitions per bus cycle.

Working Zone code [15] exploits the locality of reference that is usually present in the software programs. The proposed encoding technique partitions the address space into working zones whose starting addresses are stored in a number of registers. A bit is used to denote a hit or a miss of the working zone. When there is a miss, the full address is transmitted on the bus; otherwise, the bus is used to transmit the offset, which is one-hot coded. Additional lines are used to transmit the identifiers of the working zone. Codebook-based code [16] can be thought of as a generalized version of the Bus-Invert code. The codebook contains the set of patterns and their corresponding ID's. The patterns are chosen so that the average Hamming distance between a source word and the "best" pattern in the codebook is minimized.

Entropy-reducing Code [17] refers to a group of codes that attempt to reduce the entropy rate of the source given a fixed level of redundancy in the bus. The key idea is to compute the error between the current source word and its predicted value followed by a coding algorithm that minimizes the transition activity. The result is then sent on the bus using the Transition Signaling technique and is decoded accordingly. The rationale for this class of codes is that the power savings obtainable by encoding depend on the entropy rate of the incoming source data and on the amount of redundancy in the code. The higher the entropy rate, the lower the energy savings that can be achieved by encoding the source words for a specified level of redundant bits on the bus.

Beach code [18] analyzes the word-level correlations between source words to assign codes with small Hamming distance to data words that are likely to be sent on the bus in two consecutive clock cycles. Beach code is a subset of the entropy-reducing codes. The Beach encoder and decoder do not, however, use decorrelator and correlator blocks.

Probability-based codes [19] are generated based on a general codec architecture that uses encoder/decoder functions based on the current and previous values of the source and code words and decorrelator/correlator functions that implement a Transition Signaling scheme on the bus. These codes start with the assumption that a detailed statistical characterization of the data source is available, that is, the stationary probability distribution of all pairs of consecutive values in the input stream is known. For example, the Exact Encoding function uses an exponential table (in the bit width of the bus) that stores all possible pairs of source words and their joint occurrence probability in order to assign a minimum of transition activity codes to each pair of source words (Transition Signaling).

The key idea behind all these techniques is to reduce the Hamming distance between consecutive addresses for a sequential memory access pattern, e.g., instruction fetching or large array access. However, these schemes cannot be applied to DRAM address bus encoding because of the time-multiplexed addressing scheme used therein, which is practiced universally and due to technical and legacy issues.

B. DRAM Technology

DRAM is usually laid out in a 2-dimensional array. To identify a memory cell in the array, two addresses are needed: *row address* and *column address*. The row address is sent over the bus and latched in the DRAM decoder. Subsequently, the column address is sent to complete the address. We refer to this kind of DRAM as *conventional DRAM*. As a result, the conventional *DRAM* bus is time-multiplexed between the row and column addresses, so that the pin count for addresses is reduced by a factor of two. Because the switching activity on a DRAM bus is totally different from that of a non-multiplexed bus, we need another Gray code-like encoding scheme to minimize the switching activity for sequential memory access on a DRAM bus.

In addition to conventional DRAM, almost every modern DRAM device supports a *page mode*. In the page mode, after the first data transaction, the row address is latched and then different memory locations in the same row are read/written by sending only their column addresses. *Hyper Page* mode, i.e., *Extended Data Out* (EDO) mode, is the same as page mode, except that the Column Address Strobe (CAS) signal is overloaded with both the CAS and Data Out.

Synchronous DRAM (SDRAM), named so because it avoids the asynchronous handshaking used in conventional and page mode DRAMs, uses the system clock to strobe data. No Data-Out signal is needed. To boost the throughput, in *burst mode* DRAM, several bytes (2, 4, or more) can be read/written continuously without any handshaking signal. *Double Data Rate* (DDR) DRAM uses both the rising and falling edges to increase the bandwidth. *Rambus* DRAM (RDRAM) targets high

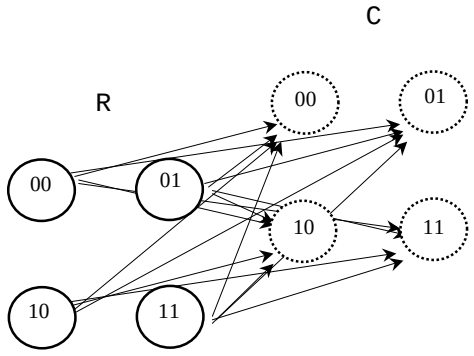
performance computer systems and has evolved three generations: Base, Concurrent, and Direct Rambus. RDRAMs are variable-length packet-switched. Because their signals are quite different from the previously mentioned DRAM devices, we exclude RDRAMs from further consideration in this paper [20].

II. PYRAMID CODE

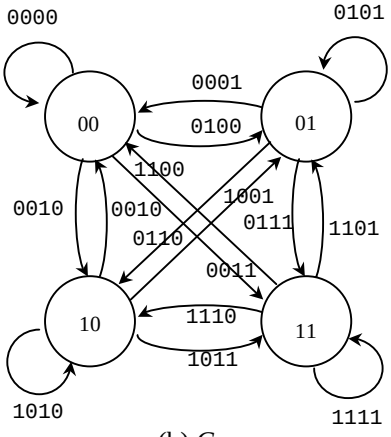
We focus on minimizing the external switching activity for a sequential access pattern in this section. The basic concepts of Pyramid code are presented.

A. Graph Representation

Without loss of generality, consider a DRAM memory space consisting of 16 (2^4) locations. Each location is identified by 4 bits, which are multiplexed on a 2-bit wide address bus. Our goal is to find a complete ordering of these 16 addresses (e.g., permutation) such that the switching activity on a multiplexed address bus is at a minimum.
We represent these addresses by a Row/Column graph G_1 in Figure 1(a). Hereafter, G_1 will be referred to as a *RC graph*. The four solid circled nodes represent the row address set R whereas the four dotted circled nodes represent the column address set C. For each pair of nodes $u \in R$ and $v \in C$, there is a *forward-edge* (u,v) representing the address $<uv>$. Each such edge has a weight equal to the Hamming distance between u and v, $H(u,v)$. This weight is called the *internal (intra-address) switching activity* of address $<uv>$. Consider two consecutive addresses $<u_1v_1>$ and $<u_2v_2>$. When transmitting these two addresses on a multiplexed bus, the *external (inter-address) switching activity* on the bus is $H(v_1, u_2)$. We define the corresponding edge (v_1, u_2) as a *back-edge* (because it goes from C to R). The back-edges are not shown in G_1. Our goal is to construct a cycle Q^* that visits all of the forward edges (i.e., all of the addresses) exactly once while minimizing the sum of the weights of the back-edges (i.e., the total external switching activity). Notice that the weight of the back-edges $<v,v>$ is zero and we will use these 0-weighted back-edges to construct the cycle.
Since R and C have the same labels, we can superimpose these two sets and get a *merged RC graph* G_2 as shown in Figure 1(b). G_2 is a complete directed graph K_4. The nodes represent row or column addresses and each edge (u,v) represents a complete address $<uv>$. These edges correspond to the forward-edges in G_1. We simply ignore the back-edges of G_1 because the 0-weighted back-edges in G_1 become 0-weighted self-edges in G_2. We claim that any Eulerian cycle on G_2 is a solution Q^*.



(a) G_1



(b) G_2

Figure 1 (a) The RC Graph. (b) The Merged RC Graph for a conventional DRAM

Theorem 1: A Eulerian cycle of graph G_2 yields a power-optimal multiplexed code for sequential addressing of the corresponding address space.

Proof. Consider solving the problem of constructing a cycle that visits all of the forward edges of G_1 exactly once while minimizing the sum of the weights of the back-edges of G_1. When we traverse a forward edge (u,v) to go from the R set to the C set, we can return to any vertex in the R set by following any back-edge that starts from v, that is, $(v,-)$. Obviously, the back edge (v,v) is the best choice since its weight is the minimum possible, that it, zero. So our problem becomes that of finding a cycle that visits all of the forward edges of G_1 exactly once while using only the zero-weighted back-edges (which can be used as many times as needed). Finding a Eulerian cycle of graph G_2 produces a power-optimal multiplexed code because along this cycle all of the forward-edges in G_1 are visited exactly once and only zero-weighted back-edges of G_1 are implicitly used. Therefore, the external switching activity becomes zero.

Sufficient and necessary conditions for a Eulerian cycle to exist on a graph are that (1) the graph is connected and (2) for every vertex the in-degree is the same as the out-degree. Clearly there are a large number of solutions for a complete graph K_i. One can apply algorithms such as depth-first search or breadth-first search to get an arbitrary solution. However, the encoding and decoding functions will have to be realized in hardware. Simple yet efficient functions are necessary for practical implementation. The functions

should not be too complex so as to offset the power saving from reduced switching activity.

B. Pyramid Code

Let's denote the Eulerian Cycle Problem on K_N as ECP_N. Figure 2 shows the solutions to ECP_1 (W_1) through ECP_4 (W_4) with edges labeled by their traversal order. W_i represents a cycle (v_0,v_1) (v_1,v_2)…. (v_N,v_0) by listing the vertices in the traversal order [v_0,v_1,…. ,v_N]. The solution to ECP_0 is trivially [0], which means a cycle of only one edge (0,0) (W_1 in Figure 2). To solve ECP_k, consider ECP_k as a bipartition $K_{k-1,1}$. For example, W_3 can be partitioned into two sets W_2 and $\{v_2\}$. Assuming ECP_{k-1} has been solved by W_{k-1}, introducing the new node v_{k-1} creates 2(k-1) cut edges plus the singular self-edge (v_{k-1},v_{k-1}). Starting from v_0, these edges can be traversed in the order [0, v_{k-1}, 1, v_{k-1}, 2, v_{k-1}, 3, …, v_{k-1}, v_{k-2}, v_{k-1}, v_{k-1}]. The formal description of this process is stated as follows:

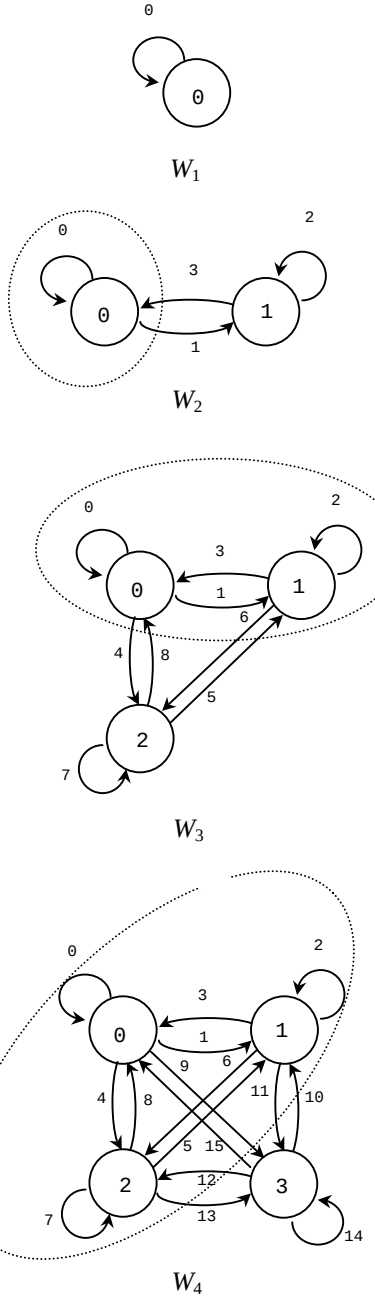


Figure 2 Examples of Eulerian cycles on the merged RC graphs

$$W_1 = [0]$$

$$W_k = W_{k-1} \,\&\, [0, \underbrace{\underbrace{k-1,1}_{1}, \underbrace{k-1,2}_{2}, \ldots, \underbrace{k-1,k-2}_{k-2}, \underbrace{k-1,k-1}_{k-1}}_{(k-1)\,pairs}]$$

where '&' denotes concatenate of two strings. For example:

$$W_2 = [0] \,\&\, [0,1,1] = [0,0,1,1]$$

$$W_3 = [00,00,01,01,00,10,01,10,10]$$

$$W_4 = [00,00,01,01,00,10,01,10,10,00,11,01,11,10,11,11]$$

The corresponding *Pyramid Code* generated from the Eulerian cycle W_4 is:

```
{0000, 0001, 0101, 0100, 0010, 1001, 0110, 1010,
 1000, 0011, 1101, 0111, 1110, 1011, 1111, 1100}
```

We name it in this way because of its topology, which looks like an i-dimensional pyramid -- W_1 is a dot, W_2 is a line, W_3 is a triangle, and W_4 is a tetrahedron. Because of our DRAM model, only W_{2j} results in the Pyramid code.

C. Pyramid I Encoding Function

In Figure 3, we use a different representation to explain the Pyramid I encoding function. A four-by-four matrix, $H_{4,4}$, represents the 16 addresses. The number inside a cell is the reverse function P^{-1} of the Pyramid I encoding function P. For example, $P(3)$=0100, so the cell in row 1, column 0 is 3. If we rotate the matrix $H_{4,4}$ by 45 degrees in clockwise direction and go through the numbers inside the cells starting from zero and proceeding in an increasing order, we observe the following pattern: (1) we traverse the cells in a top-down manner; (2) we move in the same V-shaped band until we visit all the cells; and (3) we jump back and forth on both sides of the diagonal line. For the V-shaped band corresponding to row 2 and column 2 (i.e., for numbers 4 through 8) the pattern alternates between the left stripe $h_{2,0}$ and $h_{2,1}$ and on the right stripe $h_{0,2}$, $h_{1,2}$, and $h_{2,2}$. After finishing these five cells, the next V-shaped band of row 3 and column 3, which consists of 7 cells, will be processed.

Based on this "seesaw" pattern, the encoding and decoding functions can be realized. The whole matrix $H_{N,N}$ contains N^2 elements, $h_{i,j}$. A proper sub-matrix, $H_{k,k}$, includes the left upper portion of $H_{N,N}$ (e.g., the boxed squares $H_{1,1}$, $H_{2,2}$, and $H_{3,3}$). $H_{k,k}$ has k^2 elements. Let's formally define the V-shaped band of row k and column k as $Band_k = H_{k,k} - H_{k-1,k-1}$, where the '-' sign is a set operation. Obviously, $Band_k$ has $2k$-1 elements labeled from $(k\text{-}1)^2$ to k^2-1 (e.g., 4 to 8 for $Band_3$). To encode any number x (say 6), there are three steps: (a) determine whether it is on $Band_k$ by calculating

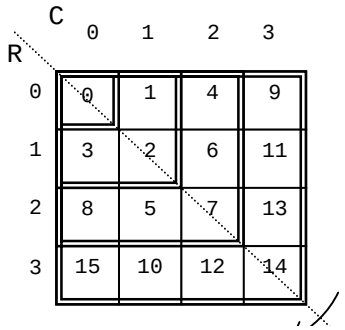


Figure 3 Matrix representation used to explain the derivation of the Pyramid I encoding function

the square root of x plus one ($\lfloor\sqrt{6}\rfloor+1=3$); (b) separate the numbers by the oddness (i.e., 5,7) or evenness (i.e. 4,6,8) of their cardinality (this is possible because the numbers on the same band are alternating on both sides of the diagonal line); (c) determine offset on the band by subtracting $(k\text{-}1)^2$ from x (6-4=2). We need to "right shift" the cells in the left stripe (5,7) by one cell (i.e., $h_{2,0}$ and $h_{2,1}$ are right shifted to $h_{2,1}$ and $h_{2,2}$, respectively). The last element on $Band_k$ (8), has to be put in the only available cell ($h_{2,0}$) because its default cell has been occupied by the second last element (7).

The Pyramid I encoding function is:

```
 1: edge (k,j,dir) {
 2:   if (dir==1)
 3:     return <k,j>;
 4:   else
 5:     if (k==j)
 6:       return <k,0>;
 7:     else
 8:       return <j,k>;
 9: }
10:
11: Pyramid_I_Encoder (x) {
12:    p = ⌊√x⌋;
13:    q = x − p^2 ;
14:    return edge(p,q/2+q%2,q%2);
15: }
```

Lines 11-15 describe the main function, which decides the band index p. Lines 1-9 calculate the exact offset on the band. Lines 2-3 decide if it is in the row (dir=0) or the column (dir=1) of the band. If it is in the column, Line 3 returns k and j as row and column, respectively. Otherwise, Line 8 swaps the row and column addresses. Line 6 handles the special case: the last cell on the band has to be "wrapped" to the first column.

Theorem 2: The Pyramid I encoding function generates a power-optimal multiplexed code for a conventional mode DRAM address bus.

Proof. The Pyramid I encoding function traces a Eulerian cycle of the corresponding merged RC-graph.

Unlike Gray code, Pyramid code is only optimal for sequential access with increasing addresses. If the sequential access pattern is decreasing, then the row and column addresses have to be swapped to preserve the code optimality.

In the implementation, we need a flooring square root function unit, an add/subtract unit and multiplexers. Unlike the square root function, the flooring square root function can be calculated in constant time by parallel N-entry table lookup. The oddness condition and shift operations can be carried out by the adder and the least significant bit of the difference between x and $(k\text{-}1)^2$. This encoder is integrated in the memory controller, so a variety of low power techniques can be applied to reduce its power dissipation overheard. The Pyramid decoding function can be found by a similar method. However, because Pyramid code is irredundant, the decoder is not needed in our proposed memory organization. It is also possible to implement a highly efficient Pyramid code incrementor and decrementor. Details are omitted here due to space limitation.

If the memory space is not too large, the encoding function can be synthesized by two or multi-level logic optimization techniques. Take 2^4 as an example, the original 4-bit address $b_3b_2b_2b_1$ will be encoded into Pyramid address $a_3a_2a_1a_0$. The Boolean functions describing the encoded bits are given below.

$$a_3 = b_2b_0 + b_3\overline{b_0}$$
$$a_2 = b_3b_1 + b_1\overline{b_0} + \overline{b_3}\,\overline{b_2}b_0 + b_3b_2\overline{b_0}$$
$$a_1 = b_2\overline{b_0} + \overline{b_3}b_2b_1 + b_3b_2\overline{b_1} + b_3\overline{b_2}b_0$$
$$a_0 = \overline{b_3}b_2 + b_3\overline{b_2}b_1 + b_3b_1\overline{b_0} + \overline{b_2}b_1\overline{b_0}$$

D. Theoretical Analysis

For binary code, the internal switching activity can be calculated as

$$SA_I(2^{2N}) = 2^N\sum_{i=0}^{N} C_i^N = 2^N(N2^{N-1}) = N2^{2N-1}.$$

The total switching activity of binary code is $N2^{2N}$, so the external switching activity is

$$SA_E(2^{2N}) = N2^{2N} - SA_I(2^{2N}) = N2^{2N-1}.$$

Pyramid code virtually eliminates all the external switching activity if the access pattern exhibits a pure sequential pattern. As a result, Pyramid code applied to a conventional DRAM bus can cut the switching activity in half.

E. Experimental Results

The purpose of our experiments is to quantitatively assess the performance of Pyramid code compared to Binary code.

We need not compare it to Gray code because Gray code performance is similar to Binary code performance on multiplexed busses.

We assume that the total memory space is 64 Kbyte (16-bit address). The address bus is 8-bits wide and row/column multiplexed. We also assume that the code address bus and data address bus are different, so the data addresses do not disturb the sequential access pattern of the code addresses. Each instruction is four-bytes long. Because the address is increased by four each time, we have to make the addresses consecutive by right-rotating them two bits before the encoding. The rotation operation has low overhead and can be integrated into the encoder. We assume that the total size of the code block is 1024 bytes. To quantitatively evaluate the effectiveness of the different degrees of address sequentiality, we divide this code block into segments of 4, 8, ..., 1024 bytes. For example, if the segment size is 8, it means that we have 128 segments with random starting addresses and within each segment we have 2 sequential addresses.

To eliminate bias due to the specific characteristics of an instruction trace, we apply a statistical sampling technique to compare Pyramid code to Binary code. More precisely, we define a sampling unit as the total number of bit transitions in a code block of 1024 instructions. We then form a sample by taking the mean of the switching activity values for 30 randomly generated sampling units. We report the expected value of the total number of bit transitions per code block of 1024 instructions by analyzing three sample results. In our experience, the sample size and number of samples is sufficient to provide high confidence (90% or higher) and low error (5% or lower) for the reported results.

Pyramid code is more efficient than Binary code when the segment size is larger than four (a segment size of four corresponds to no sequential addressing whatsoever). In practice, code segments of 8 or 16 bytes are typical. Once the segment size is larger than eight, the reduction of switching activity becomes close to 50% because Pyramid code virtually eliminates all external switching activities. We also notice that Binary code has similar internal and external switching activity. Therefore, if the access pattern exhibits a purely sequential pattern, Pyramid code will cut the switching activity by a factor of two by eliminating the external switching activity. Note that for segment size greater than 32, Pyramid code reduces switching activity by a little more than 50%. The reason is that when we go through some arbitarry segments in the memory space, the internal switching activity of Pyramid code happened to be smaller than that of Binary code. This is, however, not true for the general case and, in fact, Binary code may result in lower internal activity for a different set of examples. Notice that the magnitude of the change in external switching activity is much larger than that in interal switching activity.

Table 1

Sampling results for synthetically generated address streams

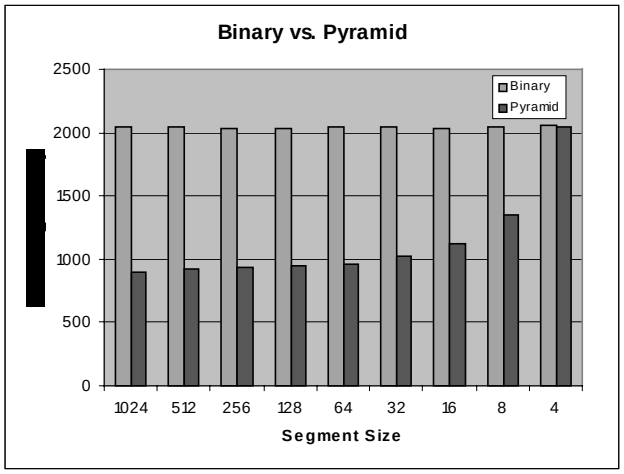
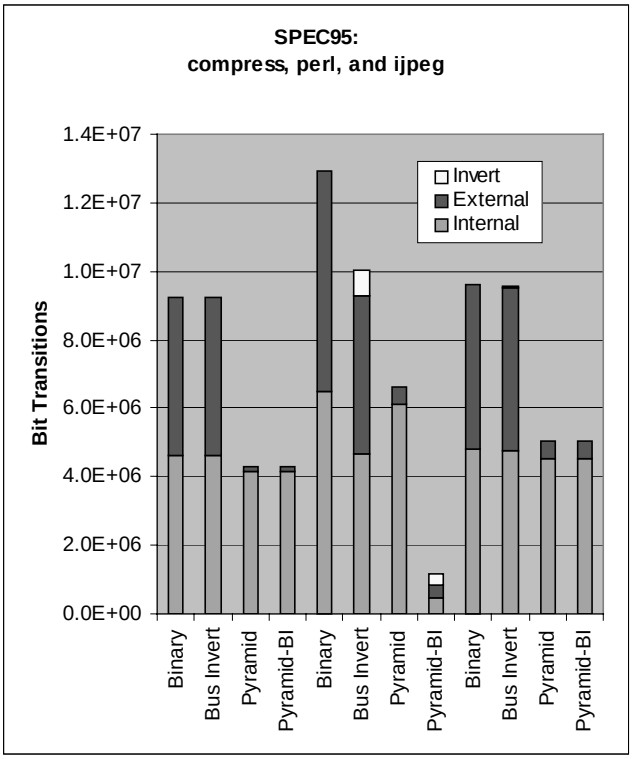


Table 2

Total bit-level transition counts for three SPEC95 benchmarks: compress, perl and ijpeg tabulated from left to right



In a second experiment, we simulated three benchmarks from the SPEC95 test suite. The three benchmarks are **compress**, **perl** and **ijpeg**. Benchmarks **compress** and **ijpeg** are representative of data intensive applications whereas **perl** is representative of control intensive applications. We simulated these benchmarks using SimpleScalar 2.0 [21] and modified the **sim-fast** memory module to filter out instruction addresses. All virtual addresses were used as physical addresses. A total of 1,000,000 addresses were collected for each benchmark. 32-bits addresses were multiplexed over a 16-bit DRAM bus. We tried four different encoding functions: Binary, Bus-Invert, Pyramid,

and Pyramid-BI (Pyramid code plus Bus-Invert signal).[1] Simulation results are presented in Table 2. Results show that for **compress** and **ijpeg** test benches, Pyramid code has the same internal switching activity as Binary and Bus-Invert codes. However, Pyramid code reduces the external switching activity on the multiplexed bus by 90%. In the case of **perl** test bench, the combination of Pyramid and Bus-Invert coding styles results in a significant improvement over Pyramid code itself. The reason is that, in this case, adding a Bus-Invert signal to the Pyramid code causes a reduction in the internal switching activity.

We assume that the virtual and physical addresses are the same. According to the random sampling experimental results, as long as the sequentiality within a page is preserved, Pyramid code can effectively reduce switching activities.

III. Reducing the Encoding Function Complexity

Pyramid code provides an asymptotic reduction in bus switching activity by a factor of 2 compared to Binary code. However, the Pyramid encoding function as proposed above is quite complex. In this section, we present a new encoding function, called Pyramid II, which has a significantly more efficient logic realization.

A. Pyramid Series

Assume that a 2*N*-bit address space is multiplexed on an N-bit bus. We use the row and column address tuple <*r*,*c*> to represent the value $r2^N+c$. Recall that Pyramid code traverses a Eulerian cycle, and that listing the nodes can represent the cycle. So we define Pyramid Series in order to describe the code. First, we define the following series:

$$E_i = 0 \cdot \prod_{j=1}^{i} (i \cdot j) \qquad E_i^{'} = 0 \cdot \prod_{j=i}^{1} (j \cdot i)$$

Symbol "·" is simply used as a delimiter between two numbers. For example:

$$E_0 = 0 \qquad E_0^{'} = 0$$
$$E_1 = 0 \cdot 1 \cdot 1 \qquad E_1^{'} = 0 \cdot 1 \cdot 1$$
$$E_2 = 0 \cdot 2 \cdot 1 \cdot 2 \cdot 2 \qquad E_2^{'} = 0 \cdot 2 \cdot 2 \cdot 1 \cdot 2$$
$$E_3 = 0 \cdot 3 \cdot 1 \cdot 3 \cdot 2 \cdot 3 \cdot 3 \qquad E_3^{'} = 0 \cdot 3 \cdot 3 \cdot 2 \cdot 3 \cdot 1 \cdot 3$$

Clearly, E_i and E'_i describe the same cycle of length $2i+1$, but in opposite directions. We will call them *forward* and *backward traversals*, respectively. The total length of $\sum_{i=0}^{k} E_i$ is $(k+1)^2$. Now, the original Pyramid I series (*P*) and Pyramid II series (*M*) for 2^{2N} can be written as:

$$P_{2^{2N}} = \prod_{i=0}^{2^N-1} E_i$$
$$M_{2^{2N}} = \prod_{i=0}^{2^{N-1}-1} (E_i \cdot E^{'}_{2^N-i-1})$$

For example,

$$P_{16} = P_{2^2 \times 2} = \prod_{i=0}^{2^2-1} E_i = E_0 \cdot E_1 \cdot E_2 \cdot E_3$$
$$= 0 \cdot 0 \cdot 1 \cdot 1 \cdot 0 \cdot 2 \cdot 1 \cdot 2 \cdot 2 \cdot 0 \cdot 3 \cdot 1 \cdot 3 \cdot 2 \cdot 3 \cdot 3$$
$$M_{16} = M_{2^2 \times 2} = \prod_{i=0}^{2^1-1} (E_i \cdot E^{'}_{2^2-i-1}) = E_0 \cdot E_3^{'} \cdot E_1 \cdot E_2^{'}$$
$$= 0 \cdot 0 \cdot 3 \cdot 3 \cdot 2 \cdot 3 \cdot 1 \cdot 3 \cdot 0 \cdot 1 \cdot 1 \cdot 0 \cdot 2 \cdot 2 \cdot 1 \cdot 2$$

Let $p(i)$ be the *i*-th number in the Pyramid series (either *P* or *M*). The encoding *f* of *x* is:

$$f(x) = \langle p(x), p(x+1) \rangle = p(x) \times 2^N + p(x+1)$$

For example, binary number 6 is encoded by *M* as

$$f(6) = \langle p(6), p(7) \rangle = p(6) \times 2^N + p(7) = 1 \times 2^2 + 3 = 7$$

P_{16} and M_{16} are listed in the last two columns of Table 3.

B. Pyramid II Encoding Function

We next explain how the Pyramid II encoding function can be efficiently implemented. First, the input number *x* is divided into three fields: *p, q,* and *s* as in Figure 4. The most significant *N*-1 bits are in field *p*. The least significant bit is *s*. The remaining bits are in field *q*. Although *p* has only *N*-1 bits, we consider *p* and *q* as *N*-bit unsigned integers, while *s* is a 1-bit number. An example is given in columns 2, 3, and 4 of Table 3.

We define a special operator *s* on a tuple or a scalar value:

$$x^s = \begin{cases} x, s = 0 \\ \overline{x}, s = 1 \end{cases}$$
$$\langle x, y \rangle^s = \begin{cases} \langle x, y \rangle, s = 0 \\ \langle y, x \rangle, s = 1 \end{cases}$$

The operation is performed only when *s*=1. For a tuple <*x*, *y*>, operator *s* swaps the two numbers and returns <*y*,*x*>. For a scalar *x*, operator *s* complements *x*. More precisely, if *x* is *s* itself, operator *s* returns (1-*s*). If *x* is *p* or *q*, the operator returns $2^N - x$. Thereby, the Pyramid II encoding function can be written as:

[1] Pyramid-BI code uses the Pyramid encoding function, but exploits a redundant Bus-Invert signal to reduce the intra-address switching activity. The manner in which the Bus-Invert signal is used is exactly the same as the way it is used in a non-multiplexed bus.

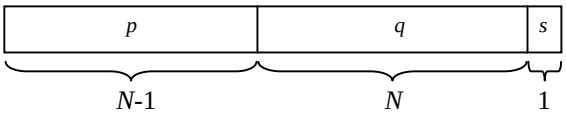


Figure 4 The *p*, *q*, and *s* fields for a 2*N*-bit number

$$\mathrm{M}(p,q,s)=\begin{cases}\left\langle p^{s},0\right\rangle^{s}, p=q\\ \left\langle q+s,p\right\rangle^{s}, p>q\\ \left\langle \overline{q+s},\overline{p}\right\rangle^{s}, p<q\end{cases}$$

In Table 3, the fourth column shows the result of comparing the p and q values. The other columns illustrate the computation steps. We next provide an intuitive explanation of why and how the M function generates the Pyramid II encoding function.

Theorem 3: The Pyramid II encoding function generates a power-optimal multiplexed code for a conventional mode DRAM address bus.

Proof. The Pyramid II encoding function traces a Eulerian cycle of the corresponding merged RC-graph.

C. Intuitive Explanation

To find the translation function from binary number x to Pyramid code $<r,c>$, we can consider it in this way: in the Pyramid series, find the x-th (r) and $(x+1)$-th (c) numbers. Assume r is the j-th (from 0) number in E_i. We know that either r or c must be i. If $r=i$, then c is either 0 or $j/2$. For

Table 3

M and P series for Pyramid I and II codes

x	*p*	*q*	*s*	*?*	*q+s,p*	$\overline{q+s},\overline{p}$	*r,c*	*M*	*P*
0	0	00	0	A	_,0		0,0	0	0
1	0	00	1	A		_,3	0,3	3	1
2	0	01	0	C		3,3	3,3	15	5
3	0	01	1	C		2,3	3,2	14	4
4	0	10	0	C		2,3	2,3	11	2
5	0	10	1	C		1,3	3,1	13	9
6	0	11	0	C		1,3	1,3	7	6
7	0	11	1	C		0,3	3,0	12	10
8	1	00	0	B	0,1		0,1	1	8
9	1	00	1	B	1,1		1,1	5	3
10	1	01	0	A	_,1		1,0	4	13
11	1	01	1	A		_,2	0,2	2	7
12	1	10	0	C		2,2	2,2	10	14
13	1	10	1	C		1,2	2,1	9	11
14	1	11	0	C		1,2	1,2	6	15
15	1	11	1	C		0,2	2,0	8	12

Pyramid I, we need to calculate the square root of x to obtain i, which is a complex operation. Pyramid II solves this problem by pairing E_i and E_{2^N-i-1}. The total length of every pair is thus 2^{N+1}, and there are 2^{N+1} pairs in total. Let $q.s$ denote the concatenation of the q and s fields. The p field indicates the pair consisting of E_p and $E'_{p'}$. The $q.s$ field indicates the position of this pair. To decide on E_p or $E'_{p'}$, (recall that the length of E_p is $2p+1$), we compare $q.s$ with $2p+1$, which is equivalent to comparing q with p. If $q<p$ (Case B), we should return the $q.s$–th number in E_p counting from the beginning of E_p; this number is obviously $q+s$. If $q>p$ (Case C), we should return the $q.s$–th number in $E'_{p'}$ counting from the end of $E'_{p'}$; this number is $q'+s'$. $q=p$ (Case A) is the special case where x is next to the boundary, and we should return p, p', or 0.

D. Experimental Results

Comparing the two Pyramid encoding functions P and M, the improvements include: (1) to calculate E_i, P uses the squared root function while M uses the N-1 most significant bits; (2) to decide the different cases, M compares only the p and q fields, but P needs to compare the results from a subtraction operation; (3) M needs the complement operation, which can be implemented efficiently; (4) because s is either one or zero, an incrementer (instead of an adder) can be used to perform the required addition operation. Although M is much simpler to implement than P in any aspect, P is independent of N. More precisely, p_i is a prefix of p_j if $i<j$ whereas M_i is completely different from M_j if $i\neq j$. This cannot be considered as a weakness of M because in practice the bit width of the memory address bus is known and fixed.

We used the ESPRESSO two-level logic minimization tool to generate a near-optimal realization of Pyramid I and II codes. The results are shown in Table 4. The savings increase with the number of bits. For a 7-bit multiplexed bus, the product term and literal count savings can be as much as 81% and 84%, respectively. Note that the logic

Table 4

Espresso Synthesis results for Pyramid I and II encoders

N	Number of Product Terms			Number of Literals		
	P	M	P/M (%)	P	M	P/M (%)
2	13	13	100	32	35	109
3	40	35	87	158	129	82
4	131	81	61	716	385	54
5	428	178	41	3003	1033	34
6	1319	377	28	11316	2587	22
7	3977	784	19	39106	6212	16

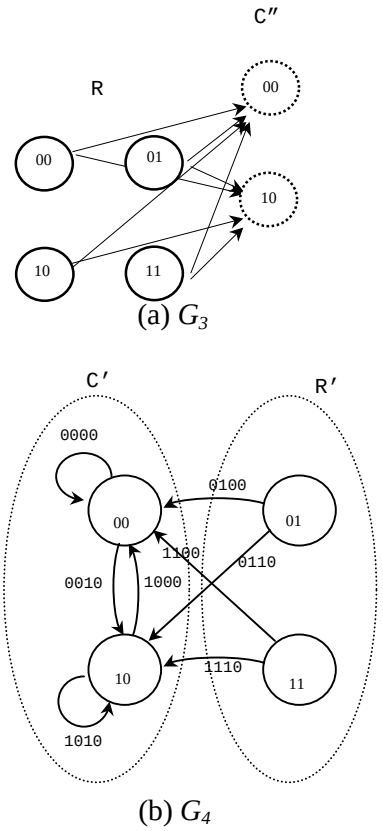


Figure 5 (a) The RC graph and (b) The merged RC graph for an aligned access with L=2

complexity of the Pyramid encoder increases rapidly with the number of bits. In practice, we only need to generate Pyramid code for the, say 8, least significant bits of the address bus.

A reasonable question at this time is what the power dissipation overhead of the Pyramid II encoder and decoder functions are. We synthesized the Pyramid-II encoder function for 8-bit and 12-bit multiplexed address busses using a 0.5-micron ASIC library from HP. We then simulated each circuit using 2^{10} and 2^{16} vectors, respectively and calculated the internal power dissipation of the encoder at a clock frequency of 100 MHz. We found this power dissipation to be less than 5% of the power dissipation on the bus (each bus bit line driver sees a 2 pF capacitive load). So, in fact, the power dissipation of the Pyramid encoder and decoder circuitry, although not negligible, is rather small. Furthermore, the encoder/decoder latency is quite small compared to the latency for bus transactions and hence the performance effect of Pyramid code is negligible.

IV. Extension to Burst Mode DRAM

A. Single-bank Burst Mode DRAM

Pyramid code can be extended to the burst mode DRAM. We assume that all the read/write accesses are of fixed length L, i.e., the addresses must be aligned at L-byte boundaries. Assuming L=2, the column set C is reduced to C'' as shown in the redrawn RC graph G_3 in Figure 5(a). The forward-edges that represent the internal switching activities are shown while the back-edges that represent the external switching activities are not shown. Our goal is to construct a cycle that visits all of the forward edges exactly once while minimizing the sum of the weights of the back-edges. We build the merged RC graph G_4 in Figure 5(b), where we have merged nodes of C'' with the corresponding nodes of R. If a Eulerian cycle of G_4 is found, we have optimally solved the problem on G_3. However, no Eulerian cycle of G_4 exists.

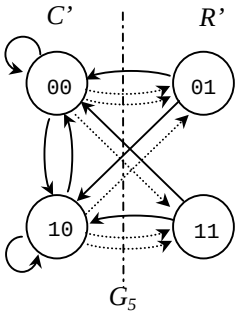


Figure 6 The merged RC graph G5 for burst mode DRAM

To construct such a cycle, we must insert some back-edges into G_4. In the merged RC graph G_4, there is a complete graph embedded on the set of nodes in C'. Consider G_5 in Figure 6 as a bipartite graph – with disjoint sets C' and R' and the cut edge set E'. E' contains all of the forward edges from R' to C'. To construct a Eulerian cycle, according to the sufficient and necessary conditions for the existence of a Eulerian cycle, we need $|R'|\times|C'|$ back-edges, or for each node v in C', we need $|R'|$ back-edges. To minimize the weighted sum of the back-edges, we choose the minimum-weight edge (v,u^*) and duplicate it $|R'|$ times. Finally, the multigraph G_5 is created as depicted in Figure 6.

Theorem 3: A Eulerian cycle of graph G_5 yields a power-optimal multiplexed code for sequential burst-mode addressing of the corresponding address space.

Proof. The proof is similar to that of Theorem 1 and follows from the construction of G_5 for the burst mode DRAM, where we add the minimum number of back-edges that are required to complete the Eulerian cycle, and furthermore, the additional back-edges have the minimum possible weight.

It is then easy to construct the *Burst Pyramid code*. For the example in Figure 6, we get the following code*:*

{0000, 0100, 0110, 1100, 0010, 1010, 1110, 1000}

The four underlined numbers are added to the original Pyramid code for C' and cause external switching activity

represented by the back-edges (00,01), (00,01), (10,11), and (10,11). The encoding function can be synthesized as:

$$a_0 = b_0$$
$$a_1 = b_3\overline{b_2} + b_2\overline{b_1}$$
$$a_2 = \overline{b_3}b_2 + b_2\overline{b_1}$$
$$a_3 = b_3b_1 + b_3b_2 + b_2b_1$$

B. *Multi-bank Burst Mode DRAM*

The memory controllers often support several memory banks. We take the non-multiplexed bank-select signals into account and thereby develop an *Interleaved Pyramid encoder* to solve the optimal encoding problem on a partially multiplexed address bus for the burst mode DRAM.

In a real micro-controller or memory controller, there are usually a set of Bank-Select signals to enable different memory chips. These signals are not multiplexed but are considered part of the address. There are two basic reasons for using multiple banks: (1) capacity - to provide the required memory size; in this case, the most significant bits are used to select banks. (2) interleaving - to reduce access time; in this case, the least significant bits are used to select banks. We treat this kind of memory organization as a partially multiplexed address bus. We use the notation *m-m-b bus* to describe a partially multiplexed bus where 2*m* bits are multiplexed and *b* bits are non-multiplexed.

By using the RC-graph, the optimal encoding for multi-bank conventional DRAM can be easily found – apply Pyramid code to the *m*-bit multiplexed sub-bus and Gray code to *b*-bit non-multiplexed sub-bus. However, we are interested in the burst mode DRAM. Because of the caches (for instruction and data), memory transactions are usually initiated in burst mode by the cache-line fill or write back events. Therefore, the burst length 2^k is programmed as the same as the cache-line size, and the starting address needs to be aligned with the cache-line size 2^k. Since only the first row and column addresses of the block are required to be sent in burst mode, we can assume that the least significant *k* address bits are always zero. Although Extended Pyramid code as described above provides the optimal encoding for a single burst mode DRAM bank, we can and should attempt to further reduce the switching activity by using multiple banks.

Figure 7 shows the organization of a 4-way Interleaved Pyramid encoder. *A* and *D* denote the high capacitance address and data busses between the encoder and the decoder, respectively. The Pyramid II encoder and decoder are employed to reduce the switching activity on this bus. For a fixed burst length of 2^B, 2^B banks are used. Instead of using the most significant bits (MSB) or the least significant bits (LSB) to select the banks, we use the encoded least significant *B* bits for the Chip-Enable inputs *E*. In this way, the banks are interleaved (although not in a regular pattern). Because the least significant *B* bits are supposed to be zero, these bits need to be converted to zero on the decoder side.

The above organization assumes that the bus width is one byte. If the bus width is 2^w, only 2^{k-w} banks are needed.

Theorem 4: The Interleaved Pyramid encoding function generates the minimum switching activity for sequential access for a *k*-way interleaved burst mode DRAM with fixed burst length of *k*.

Proof. Assume an *N-N-B* partially multiplexed bus and $k=2^B$ fixed burst length. Because the 2^B memory banks share the *N*-bit multiplexed bus, the encoder must generate all of the 2^{2N} different numbers. We create the complete RC-graph $K_{2^N}(V,E)$ to represent the 2^{2N} numbers. There are 2^B banks, so the 2^{2N} edges have to be evenly partitioned into 2^B subsets. However, the partitioning is not arbitrary. Since in the burst mode, the least significant *B* bits are fixed (in fact, they should be zero to be correctly aligned and can be so by adding inverters on the decoder side), the partitioning should depend on the column addresses. The Interleaved Pyramid encoder divides the vertices into *k* subsets $V_0, V_1 ... V_{k-1}$, and $v \in V_i$ if $f(v)=(v \bmod k)=i$. An edge (u,v) is assigned to bank *i* if $v \in V_i$. We define the *bank switching activity* SA_B on the non-multiplexed sub-bus as $SA_B(u,v) = d(f(u), f(v))$, where the distance function $d(x,y)$ is the Hamming distance between *x* and *y*. For any encoding function on $K_{2^N}(V,E)$, the total internal and bank switching activities are fixed. However, the external switching activity can change. The Interleaved Pyramid encoder generates a Eulerian cycle on $K_{2^N}(V,E)$ and has zero external switching activity. So it is an optimal encoding function.

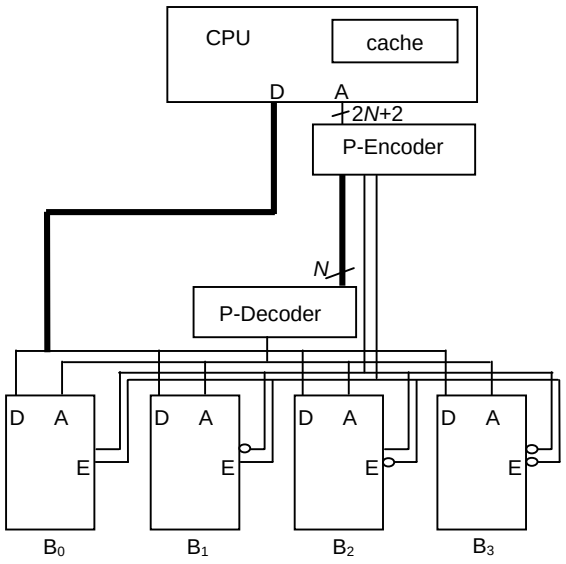


Figure 7 A 4-way Interleaved Pyramid Encoder

C. Experimental Results

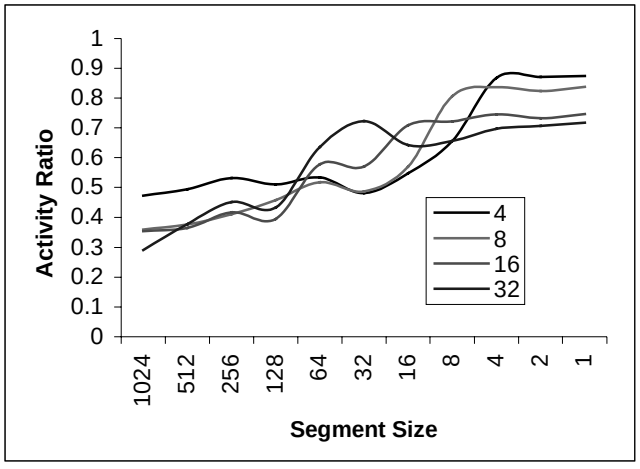
The purpose of our experiments is to quantitatively assess the performance of the Interleaved Pyramid encoder compared to a conventional *k*-bank memory organization with binary encoder. We assume that the total memory space is 64K bytes (16-bit address space). There are 4 interleaved banks. The address bus is 8-8-2 partially multiplexed. Since the code address bus and data address bus are different, the data addresses do not disturb the sequential access pattern of the code addresses. We assume that the total size of the code block is 1024 bytes. To quantitatively evaluate the effectiveness of the different degrees of address sequentiality, we divide this code block into segments of 4, 8, ..., and 1024 bytes. For example, if the segment size is 8, it means that we have 128 segments with random starting addresses and within each segment we have 8 sequential addresses.

We apply statistical sampling techniques to report the results. More precisely, we define a unit of sampling to be the total number of bit transitions in a code block of 1024 instructions. We then form a sample by taking the mean of the transition count values for 30 randomly generated sampling units. We report the expected value of the total number of bit transitions per code block by analyzing 3 sample results. In our experience, the sample size and number of samples are sufficient to provide high confidence (90% or higher) and low error (5% or lower) for the reported results.

Table 5 shows the switching activity savings for different burst lengths. The horizontal axis depicts the segment size whereas the vertical axis shows the ratio of the bus activities for the Interleaved Pyramid encoder vs. the Binary encoder. Interleaved Pyramid code always outperforms Binary code for every burst length and segment size. The switching activity saving increases (i.e., the activity ratio decreases) as the segment size increases. This is because of the increased sequentiality of the code addresses. On each curve in this figure, there is a knee at the burst length. When the segment size becomes larger than the burst length, the activity saving rate increases significantly from about 20% to 60%.

Table 5

Switching activity ratio of Interleaved Pyramid vs. Binary codes for different burst lengths: 4, 8, 16 and 32



V. Conclusions

In this paper, we presented Pyramid code, which is an irredundant power-optimal code for a level-signaling multiplexed memory bus. We formulated the problem as that of finding a Eulerian cycle on a complete or partial RC graph. We described two variants of the Pyramid encoder and showed that the Pyramid II encoder is superior to the Pyramid I encoder due to the simplicity of its function realization, which in turn minimizes the area and performance overhead of the address encoder on the memory bus. Using ESPRESSO to generate a near-optimal logic realization of the Pyramid I and II encoders, we showed a product term savings of 81% for the Pyramid II encoder compared to the Pyramid I encoder. Next, we considered single-bank and multi-bank burst mode DRAM organizations, and proposed Burst and Interleaved Pyramid code to solve the optimal encoding problem on the memory address bus. Burst and Interleaved Pyramid codes are compatible with both the Pyramid I and II encoding functions, although results were presented for the Pyramid II encoder only. Experimental results showed that Interleaved Pyramid code reduces switching activity on the bus by an average of 40% compared to the binary code.

If redundancy is allowed for encoding, we can employ the Bus-Invert signal to further reduce the memory bus switching activity. The combination of the Bus-Invert signal and Pyramid code is particularly promising as was demonstrated in the simulation results obtained for the **perl** benchmark.

References

[1] E. Macii, M. Pedram, and F. Somenzi, "High level power modeling, estimation and optimization," IEEE Trans. on Computer Aided Design, Vol. 17. No. 11, pp. 1061-1079, Nov. 1998.

[2] M. Pedram, "Power minimization in IC design: principles and applications," *ACM Trans. on Design Automation of Electronic Systems*, Vol. 1, No. 1, pp. 3-56, Jan. 1996.

[3] C. L. Su, C. Y. Tsui, and A. M. Despain, "Saving power in the control path of embedded processors," *IEEE Design and Test of Computers*, Vol. 11, No. 4, pp. 24-30, 1994.

[4] W. C. Cheng and M. Pedram, "Power-optimal encoding for DRAM address bus," *Proc. of Int'l Symp. on Low Power Electronics and Design*, pp. 250-252, July 2000.

[5] W. C. Cheng and M. Pedram, "Low power techniques for address encoding and memory allocation," *To appear in Proc. of Asia and South Pacific Design Automation Conference*, Jan. 2001.

[6] R. Murgai, M. Fujita, and A. Oliveria, "Using complementation and resequencing to minimize transitions," *Proc. of Design Automation Conf.*, pp. 694-697, June 1998.

[7] R. Murgai and M. Fujita, "On reducing transition through data modifications," *Proc. of Design, Automation and Test in Europe*, pp. 82-88, 1999.

[8] M. R. Stan and W. P. Burleson, "Bus-invert coding for low-power I/O," *IEEE Transactions on VLSI Systems*, Vol. 3, No. 1, pp. 49-58, 1995.

[9] Y. Shin, S. Chae, and K. Choi, "Partial bus-invert coding for power optimization of system level bus," *Proc. of Int'l Symp. on Low Power Electronics and Design*, pp. 127–129, Aug. 1998.

[10] S. Yoo and K. Choi, "Interleaving partial bus-invert coding for low power reconfiguration of FPGAs," *Proc. of the Sixth Int'l Conf. on VLSI and CAD*, pp. 549-552, 1999.

[11] M. R. Stan and W. P. Burleson, "Two-dimensional codes for low power," *Proc. of Int'l Symp. on Low Power Electronics and Design*, pp. 335-340, 1996.

[12] L. Benini, G. DeMicheli, E. Macii, D. Sciuto, and C. Silvano, "Address bus encoding techniques for system-level power optimization," *Proc. of Design Automation and Test in Europe*, pp. 861-866, Feb. 1998.

[13] W. Fornaciari, M. Polentarutti, D. Sciuto, and C. Silvano, "Power optimization of system-level address buses based on software profiling," *Proc. of the Eighth Int'l Workshop on Hardware/Software Codesign*, pp. 29-33, 2000.

[14] M. R. Stan and W. P. Burleson, "Coding a terminated bus for low power," *Proc. of Fifth Great Lakes Symp. on VLSI*, pp. 70–73, 1995.

[15] E. Musoll, T. Lang, and J. Cortadella, "Exploiting the locality of memory references to reduce the address bus energy," *Proc. of Int'l Symp. on Low Power Electronics and Design*, pp. 202-207, Aug. 1997.

[16] S. Komatsu, M. Ikeda, and K. Asada, "Low power chip interface based on bus data encoding with adaptive code-book method," *Proc. of the Ninth Great Lakes Symp. on VLSI*, pp. 368-371, 1999.

[17] S. Ramprasad, N. R. Shanbhag, and I. N. Hajj, "A coding framework for low-power address and data busses," *IEEE Trans. on VLSI*, Vol. 7, No. 2, pp. 212-221, June 1999.

[18] L. Benini, G. DeMicheli, E. Macii, M. Poncino, and S. Quer, "System-level power optimization of special purpose applications: the beach solution," *Proc. of Int'l Symp. on Low Power Electronics and Design*, pp. 24-29, Aug. 1997.

[19] L. Benini, A. Macii, E. Macii, M. Poncino, and R. Scarsi, "Synthesis of low-overhead interface for power-efficient communication over wide busses," *Proc. of Design Automation Conf.*, pp. 128-133, June 1999.

[20] V. Cuppu, B. Jacob, B. Davis, and T. Mudge, "A performance comparison of contemporary DRAM architectures," *Proc. of the 26th Int'l Symp. on Computer Architecture*, pp. 222-233, May 1999.

[21] D. Burger and T. M. Austin. The SimpleScalar Tool Set. Version 2.0, Technical Report CS-TR-97-1342, University of Wisconsin, Madison, June 1997.