Low Power Techniques for Address Encoding and Memory Allocation

Wei-Chung Cheng and Massoud Pedram

Dept. of EE-Systems, University of Southern California
Los Angeles, CA 90089, USA
{wccheng, massoud}@zugros.usc.edu

Abstract - This paper presents encoding techniques to optimize the switching activity on a multiplexed 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). To eliminate the external switching activity for sequential access, we propose an optimal encoding, Pyramid code, for conventional DRAM mode as well as Burst Pyramid code for burst mode DRAM. To minimize the internal switching activity, we propose Scattered Paging for both random and sequential access patterns by exploiting the built-in virtual memory mechanism, which is commonly present on modern processors.

I. Introduction

Modern electronic systems have a dichotomy of simultaneously needing to be low power and high performance. 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 the system performance and quality of service (QoS). Power-aware software compilers, dynamic power management policies, memory management and bus encoding techniques, as well as hardware design tools are needed to meet these often conflicting design requirements [8][12]. This paper focuses on the low power bus-encoding problem.

A. Address Bus Encoding

In [15], Su et al. proposed the technique of using *Gray code* to implement the program counter of a microprocessor to minimize the switching activity of sequential memory accesses. They showed that the Gray code is asymptotically optimal among all irredundant codes. The *Bus-Invert code* [14] toggles the polarity of the signals according to the Hamming distance between two consecutive data values by using an additional line on the bus. The *T0 code* [2] 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. Both Bus-Invert and T0 codes are redundant because they need one extra bit. The *Working Zone code* [10] addresses the problem that the address bus does not behave completely sequentially because the accesses to different zones are usually interleaved. The *Beach code* [1] exploits the temporal correlations for a given application. Table I lists some example codes and their total switching activities in the last row for sequential access pattern of a 16-byte memory space.

The key idea behind all of these techniques is to reduce the Hamming distance between consecutive addresses on a non-multiplexed bus. However, these schemes cannot be applied to DRAM address bus encoding because of the time-multiplexed addressing scheme used therein.

B. DRAM Technology

Dynamic RAM (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 is 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. As seen in Table I, the binary code results in very high bus switching activity.

In addition to the conventional DRAM, almost every modern DRAM device supports *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* or *extended data out mode* (EDO) is the same as page mode, except that the Column Address Strobe signal is overloaded with both the CAS and Data-Out signals.

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. Columns 6 and 7 of Table I provide examples of accessing blocks of size two in page mode and burst mode, respectively. *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 them from further

consideration in this paper. Column 8 in Table I shows the proposed *Pyramid code*, which reduces up to 50% of the bit-level switching activity in conventional DRAM bus.

The remainder of this paper is organized as follows. We provide the problem formulation for external switching activity minimization on conventional mode and burst mode DRAM address bus in Section II. Scattered Paging, a scheme for reducing the internal switching activity on DRAM address bus, is presented in Section III. Experimental results are given in Section IV. Section V concludes our work.

TABLE I
Comparison of Different Encoding Schemes

Bin	Gray	Bus-Inv	T0	Conv. DRAM	Page Mode	Burst Mode	Pyra-mid	Burst Pyra-Mid
0000	0000	0000-0	0000-0	00 00	00 00	00 00	00 00	00 00
0001	0001	0001-0	0000-1	00 01	01		00 01	
0010	0011	0010-0	0000-1	00 10	00 10	00 10	01 01	01 00
0011	0010	0011-0	0000-1	00 11	11		01 00	
0100	0110	1011-1	0000-1	01 00	01 00	01 00	00 10	01 10
0101	0111	1010-1	0000-1	01 01	01		10 01	
0110	0101	0110-0	0000-1	01 10	01 10	01 10	01 10	11 00
0111	0100	0111-0	0000-1	01 11	11		10 10	
1000	1100	0111-1	0000-1	10 00	10 00	10 00	10 00	00 10
1001	1101	0110-1	0000-1	10 01	01		00 11	
1010	1111	1010-0	0000-1	10 10	10 10	10 10	11 01	10 10
1011	1110	1011-0	0000-1	10 11	11		01 11	
1100	1010	0011-1	0000-1	11 00	11 00	11 00	11 10	11 10
1101	1011	0010-1	0000-1	11 01	01		10 11	
1110	1001	1110-0	0000-1	11 10	11 10	11 10	11 11	10 00
1111	1000	1111-0	0000-1	11 11	11		11 00	
=20	**=16**	**=28**	**=2**	**=32**	**=24**	**=16**	**=16**	**=12**

II. Pyramid Code

We focus on minimizing the external switching activity on the DRAM bus for a sequential access pattern. The basic concept of Pyramid code [4] and Burst Pyramid code will be 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 sequence of these 16 addresses (e.g. permutation) such that the switching activity is minimum. We represent these addresses by a Row/Column graph (which will be referred as *RC graph* thereafter) G_1 in Fig. 1(a). The four solid circled nodes consist of the row address set R, and the four dotted circled nodes consist of the column address set C. For each pair of $u \in R$ and $v \in C$, there is a weighted *forward-edge* (u,v) representing one of the 16 different addresses. Thus, each edge can be labeled by an address $<uv>$ with the weight equal to the Hamming distance $H(u,v)$, which reflects the *internal switching activity*. Consider two consecutive addresses $<u_1v_1>$ and $<u_2v_2>$, their *external switching activity* is calculated as $H(v_1, u_2)$. We define the corresponding edge (v_1, u_2) as a *back-edge*; these edges are not shown on G_1. Our goal is to construct a shortest cycle that includes all the 16 forward edges.

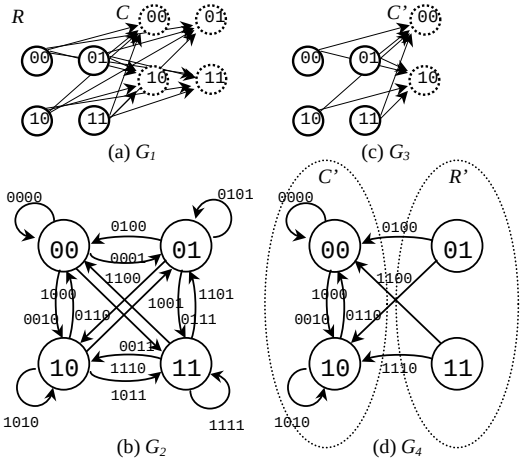

Fig. 1. (a)(c) RC Graphs. (b)(d) Merged RC Graphs.

Since R and C have the same labels, we can superimpose these two sets and obtain a *Merged RC graph* G_2 in Fig. 1(b), which is a directed complete graph K_4. By the merging, we implicitly embed a zero-weighted back-edge (v,v) in each node v. Therefore, solving the shortest cycle problem on G_1 becomes very simple on G_2. Any Eulerian cycle is an optimal solution because along this cycle all the forward-edges are connected by the zero-weighted self-loop back-edges only. Hence the external switching activity is zero. We can prove the following result.

Theorem 1. *Any Eulerian cycle on the merged RC graph G(V,E) with ordered vertex set V={0, 1, ... , 2^N-1} gives an optimal multiplexed code of the ordered set V.*

Proofs are omitted due to shortage of space.

Sufficient and necessary conditions for an Eulerian cycle to exist on a graph is that (1) the graph is connected and (2) for every vertex the in-degree is the same as the out-degree [7]. Clearly, there are a large number of such solutions in 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. Fig. 2 shows examples of ECP_1 through ECP_4 with edges labeled by the traversal order. The solution to ECP_1 is trivially [0], which means a cycle of only one edge (0,0) (W_1 in Fig. 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 (i.e. ECP for W_{k-1} exists in the sense that nodes v_0 through v_{k-2} are already covered by the Eulerian cycle). 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:

$$W_1 = [0]$$
$$W_k = W_{k-1} \,\&\, [0, \underbrace{k-1,1}_{1}, \underbrace{k-1,2}_{2}, \ldots, \underbrace{k-1,k-2}_{k-2}, \underbrace{k-1,k-1}_{k-1}]$$
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{(k-1)\,pairs}$$

where '&' denotes concatenate of two strings. Note that W_i represents a cycle by listing the vertices in the traversal order. For example:

$$W_2 = [00] \,\&\, [00,01,01] = [00,00,01,01]$$
$$W_3 = [00,00,01,01,00,10,01,10,10]$$
$$W_4 = [00,00,01,01,00,10,01,10,10,00,11,01,11,10,11,11]$$

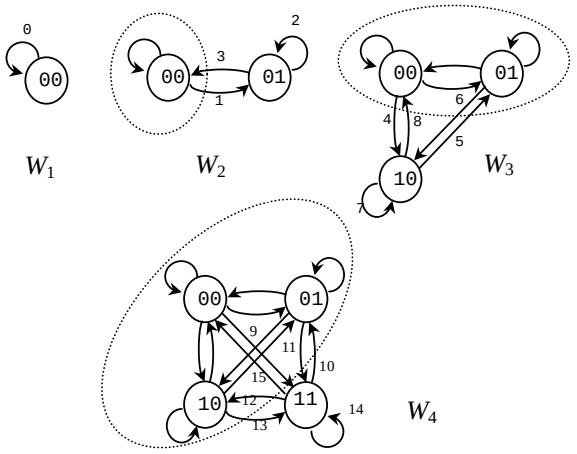

Fig. 2. Example Pyramids.

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 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. Encoding Function

In Fig. 3, we use a different representation to explain the Pyramid encoding function. The 16 addresses are represented by a four-by-four (row and column, respectively) matrix. The number inside a cell is the reverse function P^{-1} of the Pyramid encoding function P. For example, $P(3)$=0100, so the cell in row 01 and column 00 is 3. If we rotate the matrix $M_{4,4}$ by 45 degrees in clockwise direction and go through the numbers in increasing order, we observe the following pattern: (1) jump back and forth on both sides of the diagonal (2) move on the same V-shaped band and (3) traverse in a top-down manner. For the instance of 4, 5, 6, 7, and 8, the pattern goes alternatively on the left stripe [$m_{2,0}$ $m_{2,1}$] and on the right stripe [$m_{2,0}$ $m_{2,1}$], and $m_{2,2}$. After finishing these five cells, the next V-band would be the band of row 3 and column 3, which consists of 7 cells.

Based on the above regular "seesaw" pattern, the encoding and decoding functions can be implemented. The whole matrix $M_{N,N}$ contains N^2 elements $m_{0..N-1,\ 0..N-1}$. A proper sub-matrix $M_{k,k}$ includes the left upper portion of $M_{N,N}$ (e.g. the boxed squares $M_{1,1}$, $M_{2,2}$, and $M_{3,3}$). $M_{k,k}$ has k^2 elements. Define the V-shaped band of row k and column k as $Band_k = M_{k,k} - M_{k-1,k-1}$. 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) decide whether it is on $Band_k$ by calculating the square root of x plus one ($\lfloor\sqrt{6}\rfloor+1=3$); (b) because the numbers on the same band are alternating on both sides, they can be separated by the oddness (e.g. 5 and 7) or evenness (e.g. 4, 6 and 8) of their cardinality; (c) determine the offset on the band by subtracting $(k\text{-}1)^2$ from x (6-4=2). For the left odd-numbered side (5 and 7), we need to "right shift" them one cell (from $m_{2,0}$ and $m_{2,1}$ to $m_{2,1}$ and $m_{2,2}$). For the last element on the $Band_k$ (8), because its default cell is occupied by the second last element (7), we have to put it in the only available cell ($m_{2,0}$).

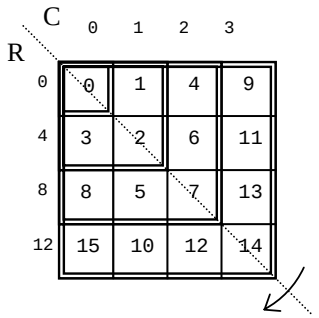

Fig. 3. $M_{4,4}$ and Seesaw Pattern

The Pyramid 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:     PyramidCode_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 addresses, 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 encoding function generates an optimal multiplexed code for a conventional mode DRAM address bus.*

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.

D. Implementation

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 of 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 [12]. 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 $b_3b_2b_1b_0$ will be encoded into Pyramid address $a_3a_2a_1a_0$. The Boolean functions describing the encoded bits are given below.

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

E. Analytical Results

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 in half the switching activity. In Table I, it is even 33% better than the page mode DRAM.

F. Burst Pyramid Code

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. In Fig. 1(c), assuming L=2, the column set C is reduced to C' as the redrawn merged RC graph G_4 in Fig. 1(d). Our goal is to find the shortest cycle including every forward-edge, but it appears that there is no Eulerian cycle for G_4.

To find the shortest cycle, we have to insert back-edges. Notice that C' is a proper subset of V. Therefore, in the merged RC graph G_4, there is a complete graph embedded on C'. Consider G_5 in Fig. 4 as a bipartite graph – with disjoint sets R' and C' and the cut edge set E'. E' contains all of the forward-edges from R' to C'. To construct an Eulerian cycle, according to the sufficient and necessary conditions for the existence of an 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 shortest edge (v,u) and duplicate it $|R'|$ times. Finally, the multigraph G_5 is created as depicted in Fig. 4. Again any Eulerian cycle on G_5 generates an optimal multiplexed code. We construct the Eulerian cycle by modifying the Pyramid code sequence for C' to include all the back-edges between C' and R'. Take Fig. 4 as an example, we get the following *Burst Pyramid code:*

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

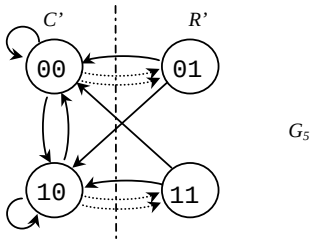

Fig. 4. The Merged RC Graph G_5 for Burst Mode.

The four underlined numbers are added to the original Pyramid code 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:

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

Theorem 3. *The Burst Pyramid encoding function generates an optimal multiplexed code for a burst mode DRAM address bus.*

In Table I, the Burst Pyramid code results in 25% reduction in the switching activity for the Burst Mode DRAM.

III. Scattered Paging

Pyramid code eliminates all external activity on a time-multiplexed bus; the internal activity is however unchanged. We next describe a technique to reduce internal switching activity by using the built-in paged virtual memory management unit.

A. Motivation

Many modern commodity microprocessors support powerful virtual memory management unit and a wide physical address bus. However, the actually installed memory is much less than the maximum memory space. This is especially true for power-sensitive portable systems. For example, popular PDAs using Motorola's DragonBall-EZ [9] with 24-bit maximum physical addressing capability usually have only 2 or 8 Mbyte installed memory [11]. We will refer to the maximum address space as MA, and the installed address space as IA in this section.

So far, our discussion uses the minimum number of signals to represent multiplexed addresses. Row and column addresses are completely overlapped (cf. Fig. 5a). On the other hand, if we double the bus width, the address overlap can be completely avoided, and thus no internal switching activity occurs (cf. Fig. 5b). If there are r overlapped bits, the internal switching activity of the overlapped portion for the complete sequential access pattern is $r2^{2r-1}$. If we reduce r by one, the internal switching activity can be reduced by 75%. The motivation is to use a larger MA to represent a smaller IA.

B. Implementation

We consider a simplified virtual memory address translation. Assume the MA is 32-bit with a page size of 4K (2^{12}) bytes. The IA is 16M (2^{24}) bytes. There are 8 bits in MA, which are unused, i.e., we can assign them arbitrarily. For example in Fig. 5(c), the 32-bit MA is multiplexed into R[15:0] and C[15:0]. We need to allocate 24-bit IA to represent the 2^{24} bytes of installed memory. The 12 least significant bits C[11:0] form the offset within the page, so they are not handled by the address translation. However because of the 8-byte block alignment requirement, C[2:0] is always 0 (assuming byte addressable; otherwise the IA would be 21-bit and we have more unused bits to manipulate), we can assign 000 to R[2:0] and eliminate the switching activities on these three bus signals. Besides the 12-bit offset C[11:0], for the other 12 bits of the IA (which actually consist of the page frame number), we need to assign them into R[15:3] and C[15:12]. We make R[15:12] and C[15:12] identical and let them represent the 4 most significant IA. The rest of the 8 (=24-12-4) IA bits go to R[11:4]. Therefore, for each address on the 16-bit multiplexed bus, we assure that there are at least 7 signals with no internal switching activity.

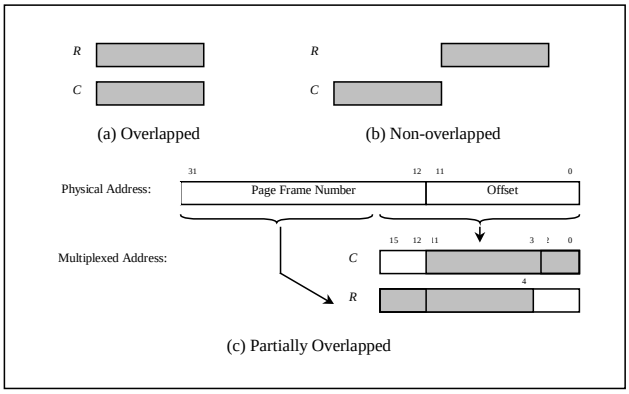

Fig. 5. Overlapped Multiplexing.

According to the above design, the physical memory space is somehow fragmented, so we need special DRAM devices with fragmented address space. Real DRAM devices, which usually have the minimum number of address signals, do not meet this requirement. Therefore, we need to implement an address decoder if we want to use real DRAM devices. The decoder is actually a multiplexer. It selects R[15:4] in the row strobe cycle, and then C[11:0] in the column strobe cycle. It turns out that the switching activity saved on the address bus is again present inside the multiplexer. However, the decoder is not part of a standard memory bus and thus can be customized with low power techniques [12].

C. Discussion

At the first glance, seven bits is not an appreciable number in comparison with the 16-bit wide multiplexed bus. However, R[15:12] are the most significant bits, which are unlikely to change in sequential access pattern. Therefore, the external switching activity can be reduced compared to the original completely overlapped bus. Furthermore, in a

virtual memory system, whenever a page fault occurs, the whole page will be paged in or out by the processor or the direct memory access (DMA) controller. Thus, the internal and external switching activities are reduced for every entry on the page even in the page mode or burst mode. Because the only difference of Scattered Paging is the layout of the physical page frames, there is no side effect on the page table organization, regardless of whether it is top-down, bottom-up or inverted. There is no impact on the performance of Translation-Lookaside Buffer (TLB) either, because the number of active pages remains the same. Nor is there any adverse effect on the cache system, either physical or virtual cache.

The switching activity saving comes from the wasted memory space and extra bus signals. The physical memory layout would be scattered because we use only those memory locations with low internal switching activity. However, application programmers and compilers usually regard the memory as a linear continuous space. If we break the memory into fragmented segments, the software environment has to be completely changed to reflect the memory fragmentation. Our approach is to use virtual memory mechanism to encapsulate this translation: the programmers or compilers still work on continuous memory space, but the physical address will be fragmented. Besides, the address encoding can be done by the built-in MMU. The only major task is to setup the virtual memory system, i.e., arrange the page table.

IV. Simulation Results

We use SimpleScalar [3] to simulate the Scattered Paging system. We use `ijpeg`, a graphic compression and decompression program, `compress`, a file compression and decompression program, and `li`, a LISP interpreter from SPEC95 benchmark suite, as our application testbench. For our input image, because its BSS segment is almost 24 Mbytes, we ignore the other segments (i.e. text, heap, and stack). We use the `sim-fast` module to perform functional simulation, and modify the `mem_access()` to capture all the memory accesses. Assuming that the installed memory is 24 Mbytes, we implement the example scattered paging system described in Section III. Table II lists the experimental results for these benchmarks. Scattered paging reduces the switching activity by 17% to 36%. In our simulation, we did not consider the paging activities between the memory and the secondary storage, which would have resulted in even higher switching activity savings.

TABLE II
Simulation Results

Switching Activity	Scattered	Conventional	Ratio
`ijpeg`	4,455k	6,901k	0.64
`compress`	393k	513k	0.76
`li`	425k	510k	0.83

V. Conclusion

In this paper, we addressed the switching activity minimization problem for conventional and burst mode DRAM address busses. We formulated the problem as that of finding an Eulerian cycle on a complete or partial RC graph. In order to make the implementation practical, we proposed efficient encoding algorithms Pyramid code and Burst Pyramid code. Experimental results show that the Pyramid code can reduce the switching activity on the bus by as much as 50% if the access pattern exhibits sequential behavior. We also proposed Scattered Paging to reduce internal and external switching activities for both sequential and random access patterns. Future work includes development of more efficient circuit realizations for the Pyramid encoder/decoder and finding other applications of the Pyramid code.

Rerferences

[1] L. Benini, G. DeMicheli, E. Macii, M. Poncino, and S. Quer. "System-level power optimization of special purpose applications: The beach solution," *ISLPED-97: ACM/IEEE International Symposium on Low Power Electronics and Design*. Monterey, CA, August 1997, pages 24-29.

[2] L. Benini, G. DeMicheli, E. Macii, D. Sciuto, and C. Silvano. "Address bus encoding techniques for system-level power optimization," *DATE-98: IEEE Design Automation and Test in Europe*. Paris, France, February 1998, pages 861-866.

[3] D. Burger and T. M. Austin. "The SimpleScalar Tool Set, Version 2.0," Tech. Rep. CS-1342, University of Wisconsin-Madison, June 1997.

[4] W. C. Cheng and Massoud Pedram. "Power-optimal encoding for DRAM address bus," *ISLPED-00: ACM/IEEE International Symposium on Low Power Electronics and Design*, pages 250-252.

[5] V. Cuppu, B. Jacob, B. Davis, and T. Mudge. "A performance comparison of contemporary DRAM architectures," *International Symposium on Computer Architecture*. (1999): 222-233.

[6] B. Jacob and T. Mudge. "Virtual Memory: Issues of Implementation," *IEEE Computer*. June 1998.

[7] D. E. Knuth. *Fundamental Algorithms*. vol. 1 of "*The Art of Computer Programming*." Addison-Wesley, 1973.

[8] E. Macii, M. Pedram and F. Somenzi. "High level power modeling, estimation and optimization," *IEEE Trans. on Computer Aided Design*, Vol. 17. No. 11 (November 1998): 1061-1079.

[9] Motorola. MC68EZ328 DragonBall-EZ Integrated Processor User's Manual.

[10] E. Musoll, T. Lang, and J. Cortadella. "Exploiting he Locality of Memory References to Reduce the Address Bus Energy," *ISLPED-97: ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 202-207, Monterey, CA, August 1997.

[11] PalmPilot. http://www.palm.com

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

[13] J. Rabaey and M. Pedram ed. *Low power design methodologies*, Kluwer Academic Publishers, 1996.

[14] M. R. Stan and W. P. Burleson, "Bus-Invert Coding for Low-Power I/O," *IEEE Transactions on VLSI Systems*, Vol. 3, No. 1 (1995), pages 49-58.

[15] 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 (1994): 24-30.