Reducing Transitions on Memory Buses Using Sector-based Encoding Technique

Yazdan Aghaghiri
University of Southern California
3740 McClintock Ave
Los Angeles, CA 90089
yazdan@sahand.usc.edu

Farzan Fallah
Fujitsu Laboratories of America
595 Lawrence Expressway
Sunnyvale, CA 94086
farzan@fla.fujitsu.com

Massoud Pedram
University of Southern California
3740 McClintock Ave
Los Angeles, CA 90089
pedram@ceng.usc.edu

Abstract
In this paper, we introduce a class of irredundant low power encoding techniques for memory address buses. The basic idea is to partition the memory space into a number of sectors. These sectors can, for example, represent address spaces for the code, heap, and stack segments of one or more application programs. Each address is first dynamically mapped to the appropriate sector and then is encoded with respect to the sector head. Each sector head is updated based on the last accessed address in that sector. The result of this sector-based encoding technique is a reduction in the number of bus transitions when encoding consecutive addresses that access different sectors. Our proposed techniques have small power and delay overhead when compared with many of the existing methods in the literature. One of our proposed techniques is very suitable for encoding addresses that are sent from an on-chip cache to the main memory when multiple application programs are executing on the processor in a time-sharing basis. For a computer system without an on-chip cache, the proposed techniques decrease the switching activity of data address and multiplexed address buses by an average of 55% and 67%, respectively. For a system with on-chip cache, up to 55% transition reduction is achieved on a multiplexed address bus between the internal cache and the external memory. Assuming a 10pF per line bus capacitance, we show that power reduction of up to 52% for an external data address bus and 42% for the multiplexed bus between cache and main memory is achieved using our methods.

Categories and Subject Descriptors: B.4.3. [Input/output and data communications]: Interconnections, Interfaces.
General Terms: Algorithms and Design.

1 INTRODUCTION
With the rapid increase in the complexity and speed of integrated circuits and the popularity of portable embedded systems, power consumption has become a critical design criterion. In today's processors, a large number of I/O pins are dedicated to interface the processor core to the external memory through high-speed address and data buses. Compared to a general-purpose high-performance processor, an embedded processor has much fewer transistors integrated on the chip. Therefore, the amount of the energy dissipated at I/O pins of an embedded processor is significant when it is contrasted with the total power consumption of the processor. It is desirable to encode the values sent over these buses to decrease the switching activity and thereby reduce the bus power consumption. An encoder on the sender side does this encoding whereas a decoder on the receiver side is required to restore the original values. For this approach to be effective, the power consumed by the encoder and the decoder has to be much less than the power saved as a result of activity reduction on the bus. Furthermore, there should be little or no delay penalty. These constraints, which are imposed on the encoder/decoder logic, limit the space of possible encoding solutions. Although numerous encoding techniques for instruction address buses have been reported ([2], [3], [4], [5], [7], [9], [10], [11], etc.), there are not as many encoding methods for data address or multiplexed address buses ([6], [9]).[1] In the case of instruction address bus encoding, high temporal correlation between consecutive addresses is exploited to decrease the number of transitions on the bus. Although sequentiality is interrupted when control flow instructions come to execution, it is still possible to encode the addresses effectively because the offset (arithmetic difference) between consecutive addresses is typically a small integer value [1]. Unfortunately, there is much less correlation between consecutive data addresses, and the offsets are usually much larger. Therefore, reducing the transitions on a data address bus in the course of bus encoding is a much more difficult task. In multiplexed address buses, compared to data address buses, there is more correlation between addresses because of the presence of instruction addresses; thus, more reduction in activity can potentially be obtained when compared to data addresses. However presence of two different address streams (i.e., instruction and data addresses) with different characteristics makes the encoding complex.

In this paper we introduce low overhead encoding methods targeting data address and multiplexed address buses. Our methods are irredundant meaning that they do not require any additional line to be added to the bus. This feature makes it possible to adopt our techniques in an existing system without making any changes to the chip pinouts and the designed printed circuit board. It will be seen that second group of our encoders are very low overhead in terms of their power consumption and delay. No time consuming operation like addition is used in them.

The rest of this paper is organized as follows. In Section 2 the related works are described. Section 3 provides the insight and a top-level view of the proposed sector-based encoding techniques. Our encoding techniques are presented in Section 4. Section 5 presents experimental results of utilizing our techniques to encode address buses and the number of gates and power consumption of our encoders. Conclusion and some future work are discussed in Section 6.

2 PREVIOUS WORK
Musoll et al. proposed the working zone method in [6]. Their method takes advantage of the fact that data accesses tend to remain in a small set of working zones. For the addresses that lie in each of these zones a relatively high degree of locality is observed. Each working zone requires a dedicated register that is used to keep track of the accesses in that zone. When a new address arrives, the offset of this address is

ISLPED'02, August 12-14, 2002, Monterey, California, USA.

1 A multiplexed address bus refers to a bus that is used for sending both instruction and data addresses.

calculated with respect to all zone registers. The address is, thus, mapped to the working zone with the smallest offset. If the offset is sufficiently small, one-hot encoding is performed and the result is sent on the bus using transition signaling (by transition signaling we mean that instead of sending the code itself we XOR it with the previous value of the bus.) Otherwise, the address itself is sent over the bus. The working zone method uses one extra line to show whether encoding has been done or the original value has been sent. It also uses additional lines to identify the working zone that was used to compute the offset. Based on this information, the decoder on the other side of the bus can uniquely decode the address.

The working zone method has also the ability to detect a stride in any of the working zones. Stride is a constant offset that occurs between multiple consecutive addresses repeatedly and can be used to completely eliminate the switching activity for those addresses. For instruction addresses, stride is the difference between the addresses of consecutive instructions. Stride is very important when instruction address encoding is tackled. In fact, the large number of sequential instructions with constant stride is responsible for the considerable transition savings that is usually seen in instruction address encoding techniques. For data addresses, stride can happen when, for example, a program is accessing elements of an array in the memory. Apart from some special cases, detecting and utilizing strides have a very small impact on decreasing the switching activity of data addresses.

The working zone method has a large area and power dissipation overhead due to the complexity of the decoder and encoder logic. In addition, it is ineffective for data address buses. This is largely due to the fact that offsets on a data address bus are often not small enough to be mapped to one-hot codes; in such a case the original address is sent over the bus, which usually causes many transitions on the bus.

Another encoding method that can be used for data addresses is the bus-invert method [7]. Bus-invert selects between the original and the inverted pattern in a way that minimizes the switching activity on the bus. The resulting patterns together with an extra bit (to notify whether the address or its complement has been sent) are transition signaled over the bus. This technique is quite effective for reducing the number of 1's in addresses with random behavior, but it is ineffective when addresses exhibit some degree of locality. To make the bus-invert method more effective, the bus can be partitioned into a handful bit-level groups and bus-invert can be separately applied to each of these groups. However, this scheme will increase the number of surplus bits required for the encoding, which is undesirable.

In [8], Mamidipaka et al. proposed an encoding technique based on the notion of self-organizing lists. They use a list to create a one-to-one mapping between addresses and codes. The list is reorganized in every clock cycle to map the most frequently used addresses to codes with fewer ones. For multiplexed address buses, they used a combination of their method and Increment-XOR [9]. In Increment-XOR, which is proven to be quite effective on instruction address buses, each address is XORed with the summation of the previous address and the stride; the result is then transition signaled over the bus. Obviously, when consecutive addresses grow by the stride, no transitions will happen on the bus. The size of the list in this method has a big impact on the performance. To achieve satisfactory results, it is necessary to use a long list. However, the large hardware overhead associated with maintaining long lists make this technique quite expensive. Furthermore, the encoder and the decoder hardware are practically complex and their power consumption appears to be quite large.

Ramprasad et al. proposed a coding framework for low-power address and data buses in [9]. Although they have introduced remarkable methods for encoding instruction addresses, their framework does not introduce effective techniques for data address and multiplexed address buses.

3 OVERVIEW

In this paper we propose three sector-based encoding techniques. All these techniques partition the address space into disjoint sectors. Each address is encoded based on the sector in which it is located. Usually the addresses in the same sector have a tendency to be close to each other; this means if we encode each address with respect to the previous address accessed in the same sector, spatial locality enables us to develop an encoding technique that results in only a small number of transitions on the bus.

To better explain this, consider two cases. In the first case, a trace of addresses, which are scattered all over the address space, is sent over a bus without any encoding. Because these addresses are dispersed, it is more likely that they have larger Hamming distances in their binary representations. In the second case, we partition the address space into two sectors so that the original trace is divided into two sub-traces based on this sectorization. In each sector, the addresses are closer to each other. If we sum up the inter-pattern transitions of these two sub-traces, this summation will be less than the total transition count for the original trace. In practice, addresses are not partitioned into two sub-traces; rather it is the function of the encoding technique to realize this "virtual separation" of addresses in the trace. This last statement reveals the key insight for the proposed sector-based encoding techniques.

Let's consider the data addresses for a memory system without cache. Each data access generated by the CPU can be either for accessing a data value in a stack, which is used for storing return addresses and local variables, or in a heap, which is used to hold the global data and dynamically allocated variables. The stack may reside in some memory segment, e.g., in the upper half of the memory, whereas the heap may reside in another memory segment, e.g., in the lower half of the memory. Let H and S denote Heap and Stack accesses, respectively. By H→S access, we mean address bus transitions that occur when the stack is accessed after a heap access. S→H, S→S and H→H are defined similarly. The number of bit transitions caused by H→S and S→H accesses are often higher than those for the S→S and H→H accesses. This is because the heap and stack sectors are usually placed very far from one another in the memory address space. Per our detailed simulations on benchmark programs, if we apply the Offset-XOR encoding technique [9] to a data address bus, S→H and H→S accesses will be responsible for almost 73% of the overall bit transitions. Now suppose we break the trace into two parts, one includes accesses to the stack, whereas the other includes the accesses to the heap. If we apply the Offset-XOR encoding to each of these two traces separately and add total transitions of each trace, then up to 61% reduction in the switching activity will be achieved with respect to the undivided trace.

A key advantage of the encoding techniques presented in this work is that they do not require any redundant bits. Obviously, in the codeword some bits are dedicated for conveying information about the sector that has been used as a reference for encoding. The remaining bits are used for encoding the offset or the difference between the new address and the previous address accessed in that sector. The value of the last access in the sector is kept in a special register called a sector head. Among our proposed techniques, the first two are only suitable when addresses accessed in two separate sectors. The first method is very general. The second method is not as general as the first one, but its implementation is much simpler and its encoder's delay is smaller . The last method is an extension of the second method, which maintains its logic simplicity and speed, yet it can support arbitrary number of sectors at the expense of a marginal hardware overhead. The problem of how to partition the address space into disjoint sectors so that addresses are evenly distributed over these sectors is a critical one. As it will be explained shortly, in the first method, the trace partitioning is dynamically changed so that the encoding method can precisely track addresses in up to two sectors. However, in the second and third methods, the partitioning is done statically. Obviously, a careless partitioning can

cause large chunks of addresses to lie in a single sector and a consequent degradation in the performance of the encoding. We will show how this scenario can be prevented by a novel sectorization of the address space.

4 ENCODING TECHNIQUES

4.1 Dynamic-Sector Encoder

Our first technique, named DS, stands for Dynamic Sector encoding. DS encoder partitions the address space into two sectors; thus, it has two different sector heads. To encode an address, its offset is computed with respect to both sector heads. The closer sector head is chosen for encoding the address. The sector heads are dynamically updated. After the codeword is computed based on the sector head that is closer to the sourceword, that sector head is updated with the value of the sourceword, i.e., one of the sector heads always tracks the addresses. A detailed explanation is provided next.

In the sequel, X and Y are assumed to be N-bit 2's-complement integers. The binary digits of X are represented by X_1 to X_N, where X_N is the MSB.

Definition 1. LSB-Inv(X) is defined as:

```
if ( X>=0 )
   LSB-Inv(X) = X
else
   LSB-Inv(X)= X  XOR (2^(N-1)-1)
```

Definition 2. Given two N-bit integers X and Y, distance of X from Y is defined as follows:

$dist(X,Y) = \{R\}_{N-1}$

$sign(X,Y) = R_N$

where R= LSB-Inv(X − Y). Note that *dist* is an (N-1)-bit integer. Notation $\{R\}_{N-1}$ denotes casting R to (N-1) bits by suppressing its MSB.

Definition 3. Given three N-bit integers A, B and X, we say X is **closer** to A when dist(X,A) is smaller than dist(X,B).

Lemma 1. As X sweeps the N-bit space, half of the time X is closer to A and half of the time it is closer to B. If X is closer to A, X +2^{N-1} will be closer to B and vice versa.

Suppose all N-bit integers are put on the periphery of a circle in such a way that 2^N-1 and 0 are next to each other. For any two integers X and Y, the length of the shortest arc between them is equal to dist(X,Y) as defined above. The direction of this arc, either clockwise or not, is shown by sign(X,Y). Based on this construction, one can easily verify Lemma 1.

Definition 4. Given two arbitrary integers A and B in the N-bit space, we define C(X,A;B) as follows:

```
S = Min {dist(X,A), dist(X,B)}    // S is an (N-1)-bit integer.

if (dist(X,A) < dist(X,B))
     M = sign(X,A)
else
     M = sign(X,B)    // M is a single bit.

if (S_(N-1) == 1)
    C(X,A;B) = NOT (M || {S}_(N-2) ) // || is the concatenation operator.
else
    C(X,A;B) = M || {S}_(N-2)    // C(X,A;B) is an (N-1)-bit integer.
```

Lemma 2. As X sweeps the N-bit space, C(X,A;B) will sweep the (N-1)-bit space. Each integer in this space is covered exactly twice: once when X is closer to A and a second time when X is closer to B.

Using Lemma 2 we explain the way the DS encoder works. We call the two sector heads SH_1 and SH_2. First, $C(X,SH_1;SH_2)$ is calculated. This is an (N-1)-bit integer. We use the MSB bit to send the sector information, which is the sector whose head was closer to the address and was used for encoding. For example, 0 can be used for SH_1 and 1 for SH_2. Lets call this bit the Sector-ID. Therefore, the DS encoder is defined as follows:

```
// DS Encoder
Codeword = (Sector-ID)|| C(X,SH1;SH2)
Update the value of the SH that is closer to X with X
```

This code is transition signaled over the bus (i.e., it is XORed with the previous value of the bus). Lemma 2 guarantees that for any arbitrary values of sector heads, the N-bit address is mapped to an N-bit integer in a one-to-one manner. As a result, it is possible to uniquely decode the numbers on the receiver side.

The LSB-Inv function used in the DS code is intended to reduce the number of 1's in the generated code since this code will be transition signaled on the bus and the number of 1's will determine the number of transitions on the bus. Note that this function is applied to 2's complement numbers to reduce the number of 1's in small negative numbers. When applied to large negative numbers, then the number of 1's is increased. In practice and on average, the LSB-Inv function is quite effective since offsets in each sector tend to be small numbers.

To obtain a better understanding of how the DS encoder works, let's ignore the function of the LSB-Inv operator. Subsequently, C(X,A;B) becomes equal to a function that calculates the offset of X with respect to either A or B, whichever is closer and then deletes its MSB. This bit deletion is necessary because one bit of the codeword is used to send the Sector-ID; therefore, only the (N-1) remaining bits can be used for the offset. Using (N-1) bits each sector head covers 2^{N-1} numbers (we consider a circular address space, i.e., 2^N=0). Half of the covered numbers are greater than the sector head and the other half are smaller (see Figure 1). Note that some addresses are covered twice, while some are not covered at all. We call the first set of addresses S_1 and the second S_2. The size of S_1 is equal to the size of S_2. Moreover, by adding 2^{N-1} or -2^{N-1} to S_1, it can be mapped to S_2. The addresses in S_1 are covered by both SH_1 and SH_2, but they are encoded with respect to the closer sector-head only. This means for each address in S_1, one code is wasted. These wasted codes can be used to encode the addresses in S_2. This is done by mapping S_2 to S_1 and encoding the numbers with

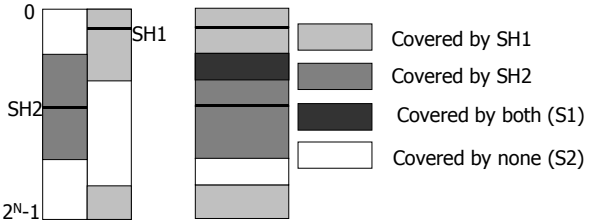

Figure 1- Address space, two sector heads and their coverage sets.

respect to the sector-head, which is *not closer*. This makes DS a one-to-one mapping.

On the receiver side, the sector is directly determined based on the MSB bit. Then, by using the value of the corresponding sector head in the receiver side (the sector heads on the sender and receiver sides are synchronized) and the remaining (N-1) bits of the codeword, the sourceword X is computed. After that, it is determined whether the

computed X is actually closer to the sector head that has been used for decoding. If true, the sourceword has been correctly calculated; otherwise, a value of 2^{N-1} should be added to X to produce the correct sourceword.

```
// DS Decoder
// Received Codeword after transition signaling is Z
U = LSB-Inv ( Z_{N-1} || 0 || {Z}_{N-2} )
if (Z_N == 0)
        X = SH_1 + U
        If (dist(X,SH_2) < dist(X,SH_1))
                X += 2^{N-1}
else
        X = SH_2+ U
        If (dist(X,SH_1) < dist(X,SH_2))
                X += 2^{N-1}

if (dist(X,SH_1) < dist(X,SH_2))
        SH_1=X
else
        SH_2=X
```

Table 1 shows an example of using DS to encode a three-bit address space. The first column denotes the original addresses (sourcewords). The two bold numbers in this column show the sector heads. The second and the third columns provide sign(X,SH) and dist(X,SH) with respect to the two sector heads. The fourth column shows the SH that has been used in calculation of $C(X,SH_1;SH_2)$. The fifth column shows $C(X,SH_1;SH_2)$. The last column shows the codewords. The MSB of the codewords shows the Sector-ID; 0 for the addresses that are encoded with respect to SH_1 and 1 for those encoded with respect to SH_2.

Table 1- An example of DS mapping, for a three-bit address space and sector heads equal to 001 and 011.

X	sign,dist (X,001)	sign,dist (X,011)	SH, SectorID	C(X,001;011)	Code word
000	1,00	1,10	001,0	10	0 10
001	0,00	1,01	001,0	00	0 00
010	0,01	1,00	011,1	10	1 10
011	0,10	0,00	011,1	00	1 00
100	0,11	0,01	011,1	01	1 01
101	1,11	0,10	011,1	11	1 11
110	1,10	0,11	001,0	01	0 01
111	1,01	1,11	001,0	11	0 11

4.2 Fixed-Sector Encoders

In this section we take a look at another set of sector-based encoding techniques that utilize fixed partitioning of address space. In each of the sectors there is a sector head that is used for encoding the addresses that lie in the sector. These techniques, which are referred to as FS, are not as general as DS, in the sense that sometimes even if consecutive addresses are far from one another, they may end up being in the same sector. Subsequently, they are encoded with respect to the same sector head and the value of the encoding totally fades away. However, the FS techniques have two major advantages over DS. The first one is the simplicity of decoder and encoder and their negligible delay overhead for the memory system and the second one is the extensibility of these methods. DS cannot be easily extended to support four sectors. If it is somehow extended, the encoder/decoder will be too complex and costly (in terms of area, delay and power overheads) to be used for low power bus encoding schemes. In contrast, as it will be seen, FS can be easily extended to support an arbitrary number of sectors. This is attractive when for example the target bus is the bus between the internal cache and the outside memory chip. Over that bus, the addresses of instructions and data blocks of multiple applications are sent to main memories, which will makeup a trace of addresses utterly scattered over the address space. A sector-based encoder needs more than two sectors to be of use for such a bus. Therefore, the importance of FS encoding techniques is realized.

4.2.1 Fixed-Two-Sector Encoder

In the Fixed-Two-Sector (FTS) encoding, the address space is partitioned into two sectors. The sectors are simply lower half and upper half of the address space. There is one sector head for each of the sectors. Each sector head consists of (N-1) bits (As the MSB is known by default). The MSB of the address or sourceword determines the sector head to be used for encoding. In addition, this MSB will be equal to the MSB of the codeword. The remaining bits are XORed with the sector head to generate the codeword. As long as the address trace is such that distant addresses lie in different sectors and, within the sectors, the addresses show some degree of locality, this technique helps reduce the transitions.

FTS encoder works as follows:

```
// FTS encoder
if (X_N ==1)
     Codeword = 1 || (SH_2 XOR {X}_{N-1})
     SH_2= {X}_{N-1}
else
     Codeword = 0 || (SH_1 XOR {X}_{N-1})
     SH_1= {X}_{N-1}
```

The codeword is transition signaled over the bus. SH_1 and SH_2 are (N-1)-bit numbers and they belong to lower half and upper half of the memory map, respectively. Therefore, the MSB of the codeword in the above equation will always be equal to the MSB of X. The simplicity of FTS comes from the fact that unlike DS, no subtraction and comparison operations are required to determine the sector head that is used. This also simplifies the decoder.

4.2.2 Fixed-Multiple-Sector Encoder

In Fixed-Multiple-Sector (FMS) encoding the address space is partitioned into multiple sectors. The number of allowed sectors is a power of 2.

Consider FTS, if all addresses lie in the lower half of the memory, then FTS encoding degenerates to that of XORing addresses with the bus which clearly leads to poor performance. FMS avoids this problem by using two techniques. First one is increasing the number of sectors. This helps to reduce the probability of having distant addresses in the same sector. Second and more important is that FMS uses a segment-based method to partition the address space, which further helps to prevent the above problem. This method is described next.

Suppose the address space is divided into 2^M sectors. If the same approach as FTS is used, the M most significant bits of the sourceword are needed to define the sectors. These bits will be the same for the sourceword and the codeword. The remaining bits in the sourceword are then XORed with the corresponding sector head to compose the codeword. However, the increased number of sectors may not be enough to evenly distribute the addresses over the sectors. Consider the main memory of a system with an internal cache. When compared to the whole address space, the main memory can be so small that it may totally reside in one of the 2^M sectors. For this reason, we propose a new technique for partitioning the address space. Now, instead of using the MSB bits, some of the center bits in the addresses are used as Sector-ID bits. Implicitly, this changes the sectors from a large contiguous section of address space to smaller disjoint (dispersed) sections. We call each of these subsections a *segment* of the sector and this type of partitioning *dispersed sectorization*.

Now consider the two different sectorization methods as depicted in Figure 2. In contiguous sectorization, the number of sectors that cover

the addresses between any two arbitrary numbers in the address space depends on the value of those boundary numbers and number of sectors. However, in dispersed sectorization, the size of the segments will also be a decisive factor to determine the number of sectors between two different addresses. Even if a subsection of the address space is small, as long as that space includes a segment from each of the sectors, addresses that lie in that space can fall in any of the 2^M sectors.

Figure 2- Comparison of contiguous versus dispersed

Suppose there are 2^M sectors in FMS. Each of the sector heads is an address bounded to one of the sectors. Consequently, M bits of each sector head are constant and known. Therefore, we only need (N-M) for storing each sector head. However, to make the following pseudo-code easier to understand we assume that sector heads are also N-bit numbers and those bits that are in the position of the sector-ID bits are all zeros. We implicitly know the sector to which each sector head belongs. The Sector-ID bits in the sourceword are used to select the correct sector head for codeword calculation and they have to be copied to the codeword exactly as they are. When these bits are XORed with corresponding zeros in the sector head, they do not change.

// FMS encoder
// 2^M sectors, 2^M Sector Heads, SH[1]...SH[2^M]
// Sector-ID bits: $X_{i+M}...X_{i+1}$ (An M-bit number)
Codeword = X XOR SH[$X_{i+M}...X_{i+1}$]
Update SH[$X_{i+M}...X_{i+1}$] with X and make the Sector-ID bits zero.

A basic question to ask is, "Which bits do we use for Sector-ID?" The number of bits defines the number and size of sectors. The location of bits defines the number and size of segments. In the sequel, we consider a bus between an internal cache and an external memory. We determine a range for the Sector-ID bits. As long as the Sector-ID bits are within that range, the reduction in switching activity will be almost the same.

We assume that Sector-ID bits are M contiguous bits in the address. Shifting the position of the Sector-ID bits to right will make the segments smaller. A main memory usually occupies a small portion of the address space. The segments should be small enough so that one segment of each sector goes into the space taken by the memory. On the other hand, the Sector-ID bits should be shifted to left to make each segment at least as large as one physical page in the memory paging system. Although consecutive pages in virtual addressing can be far from each other when they are translated to physical memory addresses, all the addresses that are in the same physical page will be very close. Suppose that multiple programs are executed. All cache misses cause requests to the external memory. Whenever a program is switched out and a new program is executed, many second level misses happen that read the code and data of the new program form consecutive blocks in physical pages. The dispersed sectorization scheme should work in a fashion to put all of these addresses in the same sector. As long as the Sector-ID bits satisfy the two aforementioned constraints, a good performance will be achieved.

5 EXPERMINETAL RESULTS

To evaluate our encoding techniques, we simulated SPEC2000 benchmark programs [13] using the simplescalar simulator [12]. The results are based on averaging over six programs named vpr, parser, equake, vortex, gcc and art. We generated three different kinds of address traces. These traces represent different memory configurations. The first two traces were generated for a memory system without an on-chip cache and are traces of data and multiplexed addresses, respectively. A data address trace includes all data accesses and assumes that data and instruction buses are separate. A multiplexed address trace includes all instruction and data addresses. The third set of traces was generated for a system with two levels of internal caches and a memory management unit that translates second level cache misses into physical addresses. The second level cache is a unified cache; therefore, addresses that miss this cache are either instruction or data addresses requesting for data and instruction blocks.

We have compared our proposed techniques with the Working Zone method with two registers or briefly WZE-2. We first show a detailed comparison of our techniques and WZE-2 when applied over the data address traces. After that we present the final results of comparison of our techniques and WZE-2 for all traces.

In Table 2 the detailed results have been shown for data address traces (no cache). For each trace we have shown the original number of transitions (Base) and the number of suppressed transitions after applying different techniques. We have also shown the percentage reduction in the total number of transitions for each set of traces and each encoding technique.

Table 2- Total suppressed transitions (in millions) and percentage savings for traces of data address (without cache).

	Base	WZE-2	DS	FTS	FMS
vpr	72.37	26.13	40.31	41.11	37.71
		36.1%	55.7%	56.8%	52.1%
parser	79.58	28.09	52.69	53.16	51.25
		35.3%	66.2%	66.8%	64.4%
equake	67.68	11.37	29.17	32.35	25.31
		16.8%	43.1%	47.8%	37.4%
vortex	87.18	17.00	34.53	39.58	41.15
		19.5%	39.6%	45.4%	47.2%
gcc	65.99	11.09	32.87	37.16	31.61
		16.8%	49.8%	56.3%	47.9%
art	83.13	12.89	41.65	49.13	45.48
		15.5%	50.1%	59.1%	54.7%
Average	0%	23%	51%	55%	51%

The same procedure has been repeated for two other sets of traces and the results are shown in Table 3. The numbers in parentheses in the FMS column shows the number of Sector-ID bits that have been used. As one can see, for data addresses and multiplexed addresses our techniques outperform the WZE-2. For the multiplexed address bus with a cache, FMS performs significantly better than the other techniques.

Table 3- Average transition saving for different techniques.

	WZ-2	DS	FTS	FMS
Data Address (No Cache)	23%	51%	55%	51%(1)
Multiplexed Address (No Cache)	47%	41%	52%	67%(3)
Multiplexed Address (Cache)	16%	19%	6%	55%(3)

Figures 2 and 3 depict the encoders for DS and FMS, respectively. The encoder for FTS has not been shown because of its similarity with the FMS encoder. The signals have been tagged with the bits they carry.

For example 32,30→1 represents bit 32 and bits 30 to 1. To better compare the overhead of these techniques, we made a comparison between the power consumption of the encoders. For this, all three encoders and decoders were designed and their netlists were generated in Berkeley Logic Interchange Format (BLIF). The netlists were optimized using SIS script.rugged and mapped to a 1.5-volt 0.18u CMOS library using the SIS technology mapper. The I/O voltage was assumed to be 3.3 volts. The address traces were fed into a gate-level simulation program called sim-power to estimate the power consumption of the encoders and decoders. The clock frequency was 50 MHZ. We also calculated the power dissipated on the bus in absence of any encoding. We assumed a 10pF capacitance per bus line. Different bus configurations were assumed for evaluation of different encoding techniques. For DS and FTS we assumed a data address bus without any internal cache. For FMS we experimented over the multiplexed bus between cache and main memory. FMS was the most efficient technique for this bus. The results are shown in Table 4. The reduced bus power shows the power after encoding. The last column shows the percentage power saving after considering the extra overhead of decoder and encoder for different techniques.

Given the fact that Working-Zone encoder needs several subtractors for calculating offsets with respect to zone registers, several comparators for choosing the zone register with the smallest offset, a subtractor and several registers and comparators for detecting the stride, and a special table for encoding the offset to a one-hot code, its overhead will be much higher than that of our sector-based encoders.

Table 4- Percentage Power Saving for Different Techniques

	Original Bus Power (mW)	Encoder Power	Reduced Bus Power (after encoding)	Power Saving
DS	13.7	0.67	6.71	41%
FTS	13.7	0.24	6.16	52%
FMS	6.7	0.41	3.01	42%

In terms of delay and area, FMS produces the best results. It only consists of four levels of logic, whereas the encoding techniques that require adding addresses or incrementing them ([2],[3],[6], etc.) need more than ten levels of logic for a 32-bit bus. The following table shows the number of gates and area required for each of the sector-based encoders.

Table 5- Comparison of the encoder hardware for the proposed techniques

	Number of gates	Area (* 1000)
DS	505	488.7
FTS	256	205.8
FMS	313	282.7

6 CONCLUSION

In this paper, we proposed a new approach toward bus encoding by sectorization of address space. The sectorization can be either dynamic or fixed. We compared different approaches in terms of power, speed and extensibility. For the multiple fixed-sector method, we introduced a technique that partitions the sectors evenly. We also showed that using our methods up to 52% power reduction for an external data address bus and 42% reduction for a multiplexed bus between internal cache and external memory can be achieved.

7 REFERENCES

[1] D. Patterson, J. Hennessy, "Computer Architecture, A Quantitative Approach", second edition, 1996.

[2] L. Benini, G. De Micheli, E. Macii, D. Sciuto, C. Silvano, "Asymptotic Zero-Transition Activity Encoding for Address Buses in Low-Power Microprocessor-Based Systems," IEEE 7th Great Lakes Symposium on VLSI, Urbana, IL, pp. 77-82, Mar. 1997.

[3] W. Fornaciari, M. Polentarutti, D.Sciuto, and C. Silvano, "Power Optimization of System-Level Address Buses Based on Software Profiling," CODES, pp. 29-33, 2000.

[4] L. Benini, G. De Michelli, E. Macii, M. Poncino, and S. Quer, "System-Level Power Optimization of Special Purpose Applications: The Beach Solution," IEEE Symposium on Low Power Electronics and Design, pp. 24-29, Aug. 1997.

[5] P. Panda, N. Dutt, "Reducing Address Bus Transitions for Low Power Memory Mapping", European Design and Test Conference, pp. 63-67, March 1996.

[6] E. Musoll, T. Lang, and J. Cortadella, " Exploiting the locality of memory references to reduce the address bus energy", Proceedings of International Symposium on Low Power Electronics and Design, pp. 202-207, Monterey CA, August 1997.

[7] M. R. Stan, W. P. Burleson, " Bus-Invert Coding for Low Power I/O", IEEE Transactions on Very Large Integration Systems, Vol. 3, No. 1, pp. 49-58, March 1995.

[8] M. Mamidipaka, D. Hirschberg, N. Dutt, "Low Power Address Encoding using Self-Organizing Lists", International Symposium on Low Power Design, Aug 2001.

[9] S. Ramprasad, N. Shanbhag, I. Hajj, "A Coding Framework for Low Power Address and Data Busses", IEEE Transactions on Very Large Scale Integration Systems, 7:212:221, 1999.

[10] Y. Aghaghiri, F. Fallah, M. Pedram, "Irredundant Address Bus Encoding for Low Power", International Symposium on Low Power Design, Aug 2001, pp 182-187.

[11] L. Macchiarulo, E. Macii, M. Poncino, "Low-energy for Deep-submicron Address Buses", International Symposium on Low Power Design, Aug 2001, pp176-181.

[12] www.simplescalar.org

[13] www.spec.org

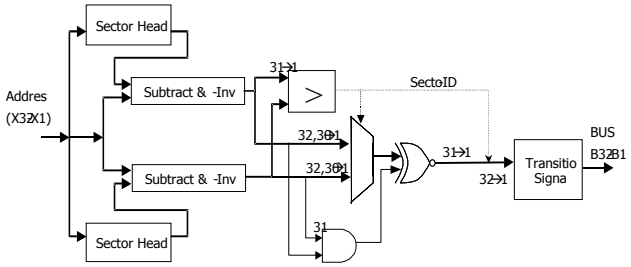

Figure 3- DS Encoder

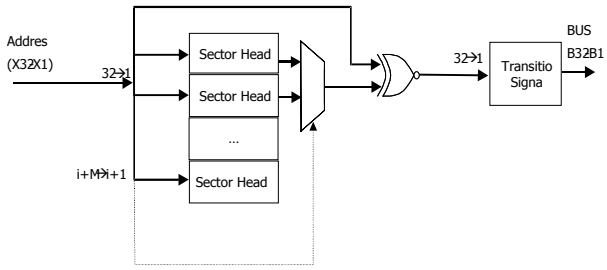

Figure 4- FMS Encoder