Transition Reduction in Memory Buses Using Sector-based Encoding Techniques

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

Farzan Fallah
Fujitsu Laboratories of America, Inc.
1240 E Arques Ave. MS 345
Sunnyvale, CA 94085
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 techniques for encoding instruction or data source words before they are transmitted over buses. The key idea is to partition the source word space into a number of sectors with unique identifiers called sector heads. These sectors can, for example, correspond to address spaces for the code, heap, and stack segments of one or more application programs. Each source word is then dynamically mapped to the appropriate sector and is encoded with respect to the sector head. In general, the sectors may be determined a priori or can dynamically be updated based on the source word that was last encountered in that sector. These sector-based encoding techniques are quite effective in reducing the number of inter-pattern transitions on the, bus while incurring rather small power and delay overheads. 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% to 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 10 pF per line bus capacitance, we show that, by using the proposed encoding techniques, a power reduction of up to 52% can be achieved for an external data address bus and 42% for the multiplexed bus between cache and main memory.

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 reducing 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 a small 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]). In the case of instruction address bus encoding, high correlation between consecutive addresses (as a result of sequential addresses) is exploited; this is to decrease the number of transitions on the bus. Although the sequential addresses are interrupted when control flow instructions are executed, it is still possible to encode the addresses effectively. This is 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. For multiplexed address buses, compared to data only, 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, the presence of two different address streams (i.e., instruction and data addresses) with different characteristics makes the encoding even more complex.

In this paper we introduce low overhead encoding methods targeting data addresses and multiplexed address buses. Our methods are irredundant, meaning that they do not require any additional lines 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 the second group of encoders, known as fixed sectors, have very low delay and power consumption overhead. The reason is that no complex operation such as 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 a detailed investigation of the overhead of encoders and decoders. The conclusion and future works 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 called zone register that is used to keep track of the accesses in that zone. When a new address arrives, the offset of the address is 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 also has the ability to detect a stride in any of the working zones. A stride is a constant offset that occurs between multiple consecutive addresses repeatedly and if detected, can be used to completely eliminate the switching activity for such addresses. For instruction addresses, stride corresponds to the offset of sequential instructions. Stride is very important when instruction address encoding is tackled. In fact, the large number of sequential instructions with constant stride is the foundation of considerable transition savings that is usually seen in instruction address encoding techniques. For data addresses, stride can occur when, for example, a program is accessing elements of an array in the memory. Except for special cases, detecting and utilizing strides has 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 can completely be ineffective with some address traces. Consider a data address bus where address offsets are not small enough to be mapped to one-hot code; 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]. The 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 one'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 of bit-level groups and a 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 absolutely 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 one's. 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 occur on the bus. The size of the list in this method has a significant 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 makes 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 addresses 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

We propose three sector-based encoding techniques. All of these techniques partition the source word space into disjointed sectors. Each source word is encoded based on the sector in which it is located. Usually the source words that are in the same sector have a tendency to be close to each other in terms of their magnitudes. This implies that if we encode each source word with respect to the previous source word accessed in the same sector, spatial locality will enable us to develop an encoding technique that would result in only a small number of bit transitions on the bus.

To elaborate on this statement, we consider two cases as follows. In the first case, a trace of source words, which are scattered all over the source word space, is sent over a bus without any encoding. Because these source words are dispersed, it is likely that they will have large Hamming distances between their binary representations. In the second case, we partition the source word space into two sectors so that the original trace is divided into two sub-traces based on this sectorization. In each sector, the source words are closer to each other and 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, source words are not partitioned into two sub-traces; it is the function of the encoder to do this "virtual separation" of source words in the trace. This statement is the key insight behind the proposed sector-based encoding techniques.

As a first example, let's consider the data addresses for a memory system without a cache. Each data access generated by the CPU may be used for either accessing a data value in a stack, which is used for storing function return addresses and local variables, or in a heap, which is used to hold 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 is often higher than those for the S→S and H→H accesses. As explained earlier, this is because the heap and stack sectors are usually placed far from one another in the memory address space. From detailed simulations performed on a large number of benchmark programs, we have observed that if we apply the Offset-XOR encoding technique [9] to a data address bus, S→H and H→S accesses will be responsible for about 73% of the entire bit transitions. Now suppose we break the trace into two parts, one includes accesses to the stack, whereas the other includes accesses to the heap. If we separately apply the Offset-XOR encoding to each of these two traces and sum up transitions for each trace, then up to 61% reduction in the switching activity will be achieved, with respect to the undivided trace.

Next consider the bus between cache and main memory. Blocks that need to be fetched from or written to the main memory belong to different processes. If two blocks belong to the same physical page, their addresses will be very close to one another. However, if they are not in the same page, then they can be significantly far from one another. The total transitions caused by two consecutive accesses, which happen to fall in the same page, tend to be much smaller than the transition counts of successive accesses in different pages. The same behavior is observed when multiple masters are communicating with different devices on the same bus. Consider AMBA bus [14] as an example. The addresses that the different masters are accessing can be uncorrelated and may cause a large number of transitions on the bus when the control of the bus is handed over.

One advantage of the encoding techniques presented in this work is that they do not require any redundant bits. Obviously, in the code word 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 (i.e., the difference between the new source word and the previous one accessed in that sector, which is kept in a special register called the *sector head*). We propose three different encoding techniques. The first two are suitable only when source words are accessed in two separate sectors. As will be

explained shortly, in the first method, the sectors are dynamically changed so that the encoding method can distinguish between source words in different sectors more generally, i.e., even if the source words get very close to each other. In the second and third methods, partitioning is done statically. For these methods, it is very important to partition the source word space into disjointed sectors such that source words are evenly distributed over all sectors.

4 ENCODING TECHNIQUES

4.1 Dynamic-Sector Encoder

The first category of encoders that we will investigate includes dynamic-sector encoders, which adaptively modify the sectorization scheme in response to the access patterns they encounter. A *sector* is a collection of source words that are encoded with respect to the same reference address, called the *sector head*. Here we examine encoding with exactly two sectors. This means the source word space is partitioned into two different sectors with one sector head in each of them; each sector is a contiguous piece of the source word space. Although it is possible to extend this method to support more than two sectors, it will not be suitable for low power bus encoding as the larger number of sector heads makes this method very complex and costly.

The dynamic-sector (DS) encoder uses two sector heads. To encode a source word, its arithmetic offset is computed with respect to both sector heads and one of the sector heads is chosen as the reference address for encoding that source word. The code word consists of two different kinds of bits, a single bit that specifies the sector head used for encoding and the remaining bits (N – 1 for an N-bit address bus) that encode the offset of the source word, with respect to the selected sector head. After the code word is computed based on the selected sector head and sent to the receiver, the sector head value is updated with the last source word. This means that the sectors are dynamically changed and that they track the source words.

To understand how N-1 bits can be used to encode an N-bit offset (i.e., the original source word minus the value of one of the sector heads), consider a circular source word space where 2^N -1 is adjacent to 0 (see Figure 1.) Consider SH_0 and SH_1 shown on the circle. An (N-1)-bit offset with respect to SH_0 will *cover* the upper portion of the circle. Similarly, the lower portion of the circle is covered by SH_1. Therefore, given the SH_0 and SH_1 shown on Figure 1(a), the entire space can be covered using an (N-1)-bit offset with respect to one of the sector heads. Now consider Figure 1(b). In this case the arcs covered by SH_0 and SH_1 intersect. There is a portion of the source word space, which is covered twice (Doubly Covered or DC) and there is a portion, which cannot be covered using an (N-1)-bit offset (Not Covered or NC). Note that, adding 2^{N-1} to a point in NC maps it to a point in DC. This fact will be used when encoding the points in NC. If a point X is in DC, it can be encoded with respect to any of two sector heads. We encode it with respect to its closest sector head. This leaves one of the potential code words unused. If

a point Y where $Y = X + 2^{N-1}$ is in NC, first 2^{N-1} is added to it. This maps it to X. Now, if the point is encoded with respect to the closest sector head, the encoding will not be one-to-one, i.e., the decoder will not be able to find out whether X or Y has been sent. Therefore, the source word is encoded with respect to the sector head, which is farthest from it (i.e., the code that has not been used.)

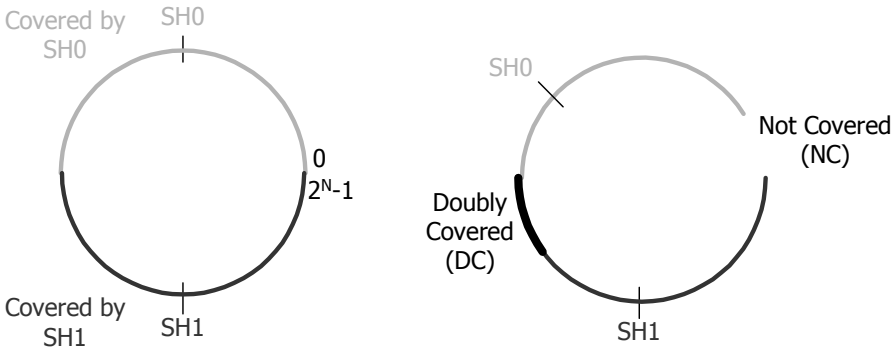

Figure 1 - Two sector heads and part of the memory space that each one covers.

In the sequel, X and A are assumed to be N-bit two's-complement integers. The bits of X are represented by X_1 to X_N, where X_N is the MSB.

Definition 1. We define dist(x, A) as

$$dist(x,A) = \begin{cases} x - A & if\ \ 0 \le x - A \le 2^{N-1} - 1 \\ A - x - 1 & if\ \ 0 \le A - x - 1 \le 2^{N-1} - 1 \end{cases}$$

We say X is **closer** to A than B when dist(X,A) is smaller than dist(X,B). Take note that dist(X,A) is a non-negative number.

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

Proof. Proving that both X and $X+2^{N-1}$ cannot be closer to A will be sufficient since it will partition the source-word space into two subgroups; one of them closer to A and the other closer to B. For any two N-bit numbers X and A as depicted in Figure 2, it is easy to verify that:

$$dist(x,A) + dist(x + 2^{N-1}, A) = 2^{N-1} - 1 = \text{constant}.$$

Therefore, for any arbitrary numbers A and B, we will have:

$$dist(X,A) + dist(X + 2^{N-1}, A) = dist(X,B) + dist(X + 2^{N-1}, B) \Rightarrow$$

$$dist(X,A) \geq dist(X,B) \Leftrightarrow dist(X + 2^{N-1}, A) \leq dist(X + 2^{N-1}, B)$$

and the proof is complete. ■

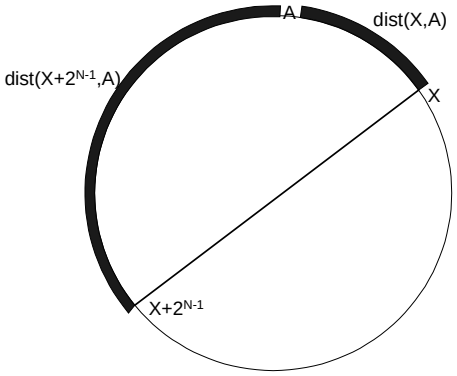

Figure 2 - Graphical representation of distance (dist) function relationship.

Note that dist(X,A) is equal to the length of the shortest arc between A and either X or X+1. If A and B are both even or both odd, dist(X,A) and dist(X,B) will never be equal and X will be closer to one of A or B. This simplifies the encoding and decoding. Thus, we enforce the requirement that both sector heads must always be odd numbers.

Definition 2. We define LSB-Inv(X) as follows:

if ($X \geq 0$) then LSB-Inv(X) = X

else LSB-Inv(X)= X XOR (2^{N-1}-1) // if X is negative, invert all bits except the MSB

It is easy to verify that dist(x, A) = {LSB-Inv (X – A)} $_{N-1}$ where $\{R\}_{N-1}$ refers to casting R to (N-1) bits.

Definition 3. We define C(X,A;B) as follows:

if (dist(X,A) < dist(X,B)) then C(X,A;B) = LSB-Inv ($\{X - A\}_{N-1}$) // LSB-INV is done in (N-1)-bit space

else C(X,A;B) = LSB-Inv ($\{X - B\}_{N-1}$)

Theorem. When 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.

Proof. From the previous lemma, it is clear that C(X,A;B) is calculated with respect to A for half of the source words and with respect to B for the other half. Since LSB-Inv is a one-to-one mapping of source-words in the (N-1)-bit space, it does not have any impact on the proof and the emphasis is on $\{X-A\}_{N-1}$ and $\{X-B\}_{N-1}$.

For every (N-1)-bit number X, either X_1=LSB-Inv^{-1}(X)+A or X_2=LSB-Inv^{-1}(X)+A+2^{N-1} is closer to A and will generate X. We call it X_{A1} and we have $\{X_{A1}-A\}_{N-1}$= X . Similarly, there is another number X_{B2}, which will generate X and is closer to B. X_{A1}

and X_{B2} are distinct numbers because one of them is closer to A and the other is closer to B. Therefore, for every (N-1)-bit number X, we have found two distinct numbers (X_{A1} and X_{B2}) such that applying function C to them will generate X and the proof is complete. ■

Using the theorem, we explain how the DS encoder works. We call the two sector heads SH_0 and SH_1. First, $C(X,SH_0;SH_1)$ is calculated. This is an (N-1)-bit integer. We use the MSB bit to send the sector identifier, which represents the sector whose head was closer to the source word and has been used for encoding. For example, 0 can be used for SH_0 and 1 for SH_1. We call this bit the *Sector-ID bit*. Therefore, the DS encoder is defined as follows:

// DS Encoder	
Code word = (Sector-ID) || $C(X,SH_0;SH_1)$	// || denotes the concatenation operator
if ($dist(X,SH_0) < dist(X,SH_1)$) then $SH_0 = Odd(X)$	// Odd(X) is the smallest odd number greater than or equal to X
else $SH_1 = Odd(X)$	// Set the sector head to current source word

The resulting code is transition signaled over the bus (i.e., it is XOR'ed with the previous value of the bus.) The above theorem guarantees that for any arbitrary values of sector heads, the N-bit source word 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.

Let's take a closer look at how this one-to-one mapping is put together. Consider an arbitrary source word X in the DC set. Obviously, $X+2^{N-1}$ is in the NC set. Assume that X is closer to SH_0 so that it will be encoded with respect to SH_0. We wish to assign a code word for $X+2^{N-1}$. The reasonable candidate is the code word that would have been generated for X, if it had been encoded with respect to SH_1. On the other hand, consider that we forget for the moment that $X+2^{N-1}$ is not covered by any of the two sector heads. From the above theorem, $X+2^{N-1}$ is closer to SH_1. So, if we simply use the rule that every source word should be encoded with respect to the sector head that is closest to it, then $X+2^{N-1}$ will be encoded with respect to SH_1. Now we know that the encoding of X and $X+2^{N-1}$ with respect to SH_1 will be exactly the same, because both of them will have the same offset with respect to SH_1. In formula form:

$$\text{LSB-Inv}\left(\{X - SH_1\}_{N-1}\right) = \text{LSB-Inv}\left(\{X+2^{N-1} - SH_1\}_{N-1}\right)$$

Consequently, by adopting the simple and straightforward rule of "encoding a source word with respect to its closest sector head," we will be using all of the unused code words for source words in the DC set to encode the source words in the NC set. This process is equivalent to mapping the NC set to the DC set (by inverting the MSB bit) and encoding the source words with respect to the sector head that is *farthest* from them, which will clearly result in a one-to-one mapping. Therefore, our general rule is to measure the distance from a source word to both sector heads and encode the source word with respect to the closest

sector head. As a final step, LSB-Inv is applied to the lower (N-1) bits of the word to reduce the number of one's in small negative offsets [10]. When LSB-Inv applied to large negative numbers, then the number of one's in the word will increase. In practice, *on average* the LSB-Inv function is quite effective in reducing the number of one's in the word, since offsets in each sector tend to be small integer numbers.

On the receiver side, an XOR operation is first required to reverse the effect of transition signaling and extract the code word. The sector is easily determined based on the MSB. Next, by using the value of the corresponding sector head in the receiver side (note that the sector heads on the sender and receiver sides are synchronized) and the remaining (N-1) bits of the code word, the source word X is computed. After this step, it is determined whether the computed X is actually closer to the sector head that has been used for decoding. If this is true, the source word has been correctly calculated; otherwise, a value of 2^{N-1} will be added to X to produce the correct source word (note that adding 2^{N-1} is equal to inverting the MSB.).

```
// DS Decoder
// Z is the received code word after transition signaling
U = Z_{N-1} || ( LSB-Inv ( {Z}_{N-1}))  // This is equivalent to applying LSB-Inv first and then doing sign-extension
if (Z_N == 0) then
        X = SH_0 + U
        If (dist(X,SH1) < dist(X,SH0)) then X += 2^{N-1}
else
        X = SH_1+ U
        If (dist(X,SH0) < dist(X,SH1)) then X += 2^{N-1}
if (dist(X,SH_0) < dist(X,SH_1)) then SH_0=Odd(X)
else SH_1=Odd(X)
```

Table 1 shows an example of using DS to encode a three-bit source word space. The first column denotes the original source words. The two underlined numbers in this column show the sector heads (SH_0 is 001 and SH_1 is 011.) The second and third columns provide the distance with respect to the two sector heads. The fourth column shows the SH that should be used in the calculation of $C(X,SH_0;SH_1)$, which is determined by comparing the distances to the two sector heads. The fifth column shows the offset of X to the closest sector head. The next column is $C(X,SH_0;SH_1)$, which is calculated by casting the previous column to two bits and then applying the LSB-Inv function. The last column shows the code words before transition signaling. The MSB of the code words shows the Sector-ID, i.e., it is zero for the addresses that are encoded with respect to SH_0 and one for those encoded with respect to SH_1.

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

X	dist(X,001)	dist(X,011)	Sector-ID, SH	X - SH_i	C(X,001;011)	Code Word
000	00	10	0,001	111	10	0 10
001	00	01	0,001	000	00	0 00
010	01	00	1,011	111	10	1 10
011	10	00	1,011	000	00	1 00
100	11	01	1,011	001	01	1 01
101	11	10	1,011	010	11	1 11
110	10	11	0,001	101	01	0 01
111	01	11	0,001	110	11	0 11

Table 2 shows how the values received on the bus are decoded in the receiver. The first column shows the bus value and the next column shows the code word that is extracted form the bus by XORing consecutive values on the bus. The third column shows the sector head that should be used for decoding based on the Sector-ID bit of the code word. The next column shows U+SH (refer to the decoding algorithm). If X (source word) is covered by at least one of the code words, U+SH will be equal to X. Otherwise, its MSB is inverted to compute X. Finally, the last two columns show the updated value of sector heads after each decoding. Initial values of the bus and sector heads have been shown in parenthesis at top of their respective columns.

Table 2 - An example of the DS decoder.

Bus (000)	Codeword	Sector-ID, SH	U+SH_i	X	SH_0 (001)	SH_1 (011)
000	000	0,001	001	001	001	011
110	110	1,011	010	010	001	011
111	001	0,001	010	110	111	011
101	010	0,111	110	110	111	011

The DS encoding method can be extended to support more sectors. The resulting encoding/decoding function will be rather slow and power consuming and will not be suitable as a low power bus encoding technique. However, the encoding is an interesting function that may be useful in other applications. In this extension, there will be no restriction on the number of sectors except that it should a power of two. Therefore, more than one bit may be used for the Sector-ID. The rest of the bits will contain the offset information with respect to one of the sector heads. Therefore, if a stream of source words is properly sectorized, this multi-sector dynamic sectorization method can significantly reduce the transition activity of the stream.

Another technique, which can help to further reduce the transition count, is to minimize the transitions associated with Sector-ID bit(s). Sometimes it is possible to predict the Sector-ID; therefore it will be enough to send only the miss-prediction information. In this case, transition of the Sector-ID bits can be significantly reduced. This technique will have a significant impact if the number of sectors is high.

4.2 Fixed-Sector Encoders

In this section we take a look at another set of sector-based encoding techniques that utilize static partitioning of the source word space. By exploiting a fixed sector scheme, the encoding and decoding functions become much simpler. These techniques, which will collectively be referred to as FS encoders, are not as general as the DS encoders in the sense that two consecutive source words, which are far from each other, may be assigned to the same sector. In such a case, they will be encoded with respect to the same sector head; therefore, a large number of bit-level transitions may occur when the code words are sent over the bus. However, the FS encoders have two major advantages over the DS encoders:

1. Simplicity - The encoder functions are simple, and consequently, have negligible delay overhead,
2. Scalability - If DS is scaled to support more than two sectors, the encoder/decoder will become too complex and costly (in terms of area, delay and power consumption) to be used for low power purpose. In contrast, as it will be seen, FS can easily be scaled to support an arbitrary number of sectors. This is attractive, for example, when the target bus is the bus between the internal cache and the outside memory chip or a bus shared by multiple masters and slaves. On the bus between the cache and the main memory, the addresses of instructions and data blocks of multiple applications are sent to the main memory, which will make a trace of addresses scattered over the address space. On a shared bus between multiple masters and slaves, although addresses generated during communication between a particular master-slave pair tend to be highly correlated, these addresses can be quite uncorrelated when considering communications among different master-slave pairs. A sector-based encoder will thus need more than two sectors to be effective for such buses.

4.2.1 Fixed Two-Sector Encoder

A Fixed Two-Sector Encoder is the simplest two sectors scheme. The sectors are defined as the lower half and the upper half of the source word space. There is one sector head for each sector. Since the MSB is constant and known for each sector, we use only (N-1) bits to show the sector head. The MSB of the source word determines if the sector head is to be used for encoding. In addition, the MSB is copied to the MSB of the code word. The remaining bits are obtained by XORing the sector head with the source word. To do this, we have to cast the sector head to an N-bit number; more precisely, we concatenate a zero with it at the left.

```
// FTS encoder
if (X_N == 0) then
    Code word = (0 || SH_0) XOR X      // concatenating SH_0 with a zero and then XORing it with X
    SH_0= {X}_N-1
else
    Code word = (0 || SH_1) XOR X
    SH_1= {X}_N-1
```

The code word is transition signaled over the bus. SH_0 and SH_1 are (N-1)-bit numbers and they are in the lower half and upper half of the memory map, respectively. The simplicity of FTS comes from the fact that, unlike DS, no subtraction or comparison is required to determine the sector head to be used. This simplifies both the encoder and the decoder. Next we will see how FTS can be modified to support more sector heads.

4.2.2 Fixed Multi-Sector Encoder

In Fixed Multi-Sector (FMS) encoding the source word space is partitioned into multiple sectors, where the number of sectors is a power of 2. Consider FTS encoding when all source words lie in the lower half of the memory. Clearly the FTS encoding degenerates to XORing source words over the bus, which performs poorly. FMS overcomes this potential weakness by applying two methods:

1. Increasing the number of sectors. This helps to reduce the probability of having distant source words in the same sector.
2. Using segments to partition the source word space.

A detailed description of FMS follows. Suppose the source word space is divided into 2^M sectors. If the same approach as FTS is used, the M most significant bits of the source word are required as Sector-ID bits. These bits will be the same for the source word and the code word. The remaining bits in the source word are then XORed with the corresponding bits of the sector head to compose the code word (similar to FTS, sector-heads are (N-M) bits each and should be concatenated with M zeros before the XOR operation.) However, the increased number of sectors may not be sufficient to achieve a uniform distribution of the source words over the sectors. Consider the main memory of a system with an internal cache. Compared to the whole address word space, the size of the main memory is so small that it may reside in one of the 2^M sectors. For this reason, we propose a new technique for partitioning the source word space. Instead of using the MSB bits, some of the intermediate bits of the source words are used as Sector-ID bits. This changes the sectors from large contiguous sections to smaller disjointed

(dispersed) sections. We call each of these subsections a *segment* of the sector and this type of partitioning *dispersed sectorization*.

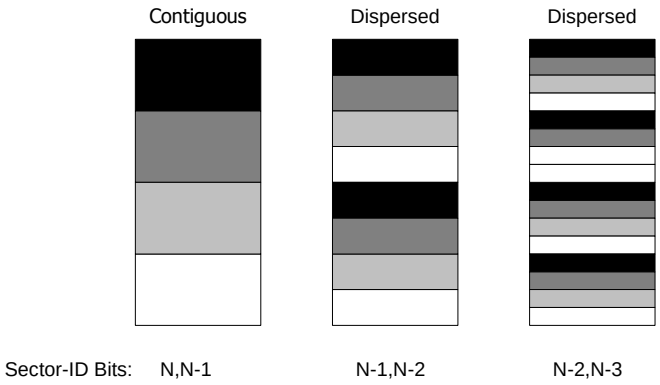

Figure 3 - Comparison of contiguous versus dispersed sectorization.

Now consider the two different sectorization methods as depicted in Figure 3. In contiguous sectorization, the number of sectors that cover the source words between any two arbitrary numbers in the source word space depends on the value of those boundary numbers and the number of sectors. However, in dispersed sectorization, the size of the segments will also be a factor in determining the number of sectors between two different source words. Even if a subsection is small, as long as that space includes a segment from each of the sectors, source words that lie in that space can fall in any of the 2^M sectors.

Suppose there are 2^M sectors in FMS. Each of the sector heads is a source word bounded to one of the sectors. Consequently, M bits of each sector head are fixed and known. Therefore, we only need (N-M) for 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 zeros. The sector to which each sector head belongs is known based on the sector head index. The Sector-ID bits in the source word are used to select the correct sector head for code word calculation and they are copied to the code word like the FTS encoder. We use these bits to arrive at the corresponding sector head and XOR the source word with that sector head to generate the code word. The Sector-ID bits in the source word are XORed with corresponding zero's in the sector head. Hence, they do not change.

```
// FMS encoder
// 2^M sectors, 2^M Sector Heads, SH[0]…SH[2^M-1]
// Sector-ID bits: X_{i+M}…X_{i+1} (An M-bit number)
Code word = 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 fundamental question is what are the bits we should use for the Sector-ID? The number of bits defines the number and the size of the sectors. The location of the bits defines the number and the size of the segments. Notice that these bits can be chosen from non-consecutive locations in the word. The answer to the above-mentioned question depends on the characteristics of the source word. In the sequel, we consider a bus between an internal cache and an external memory. For such a system, based on our simulations, we can state that there is an optimum range for the Sector-ID bits. As long as the Sector-ID bits are within that range, their exact location does not matter. Assume that Sector-ID bits are M contiguous bits in the source word. As depicted in Figure 3, shifting the position of the Sector-ID bits to the 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 at least one segment of each sector is included in the memory space. This guarantees that all sectors will be used for the encoding. On the other hand, the Sector-ID bits should be shifted to the left to make each segment at least as large as one physical page in a memory paging system. Although consecutive pages in virtual address space can be far from each other, they may be translated to close physical memory addresses. However, as long as the addresses are within the same virtual page, they will be mapped to the same physical page. Therefore, they will remain just as close as they were before translation. Suppose multiple programs are executed in the system. All cache misses generate requests to the external memory. Every time execution of a program is stopped and another program is executed, many cache misses happen. These misses are due to reading the code and data of the new program from consecutive blocks in physical pages. The dispersed sectorization scheme should place all of these source words in the same sector. This is why it is beneficial for the segments to be larger than the physical memory pages. As long as the Sector-ID bits satisfy the two aforementioned constraints and remain within the corresponding bounds, high performance can 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 sets 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 only the data accesses and assumes that data and instruction buses are separate. A multiplexed address trace includes both instruction and data addresses. The amount of correlation in multiplexed address traces is much higher than in data address traces because of the instruction 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 WZE-2 for short [6]. First, we present a detailed comparison of our techniques and WZE-2 when applied to the data address traces. Next, we present the end results of similar evaluations of our techniques and WZE-2 over different sets of traces.

In Table 3, 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 number of transitions after applying different techniques. We have also shown the percentage reduction in the total number of transitions for each trace and each encoding technique and average for all traces.

Table 3 - Total transitions (in millions) and percentage savings for traces of data address (no cache).

	Base	WZE-2	DS	FTS	FMS
vpr	72.37	46.24	32.06	31.26	34.66
		36.1%	55.7%	56.8%	52.1%
parser	79.58	51.49	26.89	26.42	28.33
		35.3%	66.2%	66.8%	64.4%
equake	67.68	56.31	38.51	35.33	42.37
		16.8%	43.1%	47.8%	37.4%
vortex	87.18	70.18	52.65	47.6	46.03
		19.5%	39.6%	45.4%	47.2%
gcc	65.99	54.9	33.12	28.83	34.38
		16.8%	49.8%	56.3%	47.9%
art	83.13	70.24	41.48	34	37.65
		15.5%	50.1%	59.1%	54.7%
Average Savings	0%	23%	51%	55%	51%

Similar experiments were performed on the two other sets of traces. The results are shown in Table 4. The numbers in parentheses in the FMS column show the number of Sector-ID bits that have been used. As one can see, our techniques outperform the WZE-2. For the multiplexed addresses with cache, FMS performs significantly better than other techniques.

Table 4 - Average transition savings 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 (w. Cache)	16%	19%	6%	55%(3)

Figure 4 and Figure 5 depict the encoders for DS and FMS, respectively. The encoder for FTS has not been shown because of its similarity to the FMS encoder. In these figures, the signals have been tagged with the bits they carry. For example, 31→1

represents bits 31 to 1. To evaluate the overhead of the techniques, we estimated the power consumption of the encoders. For this, we designed all encoders in Berkeley Logic Interchange Format (BLIF) and used SIS to generate their netlists. The netlists were optimized using SIS *script.rugged* and mapped to a 1.5-volt 0.18μm 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* in order to estimate the power consumption of the encoders and decoders. The clock frequency was 50 MHz. We then calculated the power dissipated on the bus in the absence of any encoding. We assumed a 10pF capacitance per bus line. Then, we calculated the amount of bus power saved as a result of applying different encoding techniques. For DS and FTS we used a data address bus with no cache. For FMS we experimented over the multiplexed bus between a cache and the main memory. FMS was the most efficient technique in this case. The experimental results are reported in Table 5.

Note that the 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 finally a special table for encoding the offset to a one-hot code. This means its overhead is much higher than our sector-based encoders.

Table 5 - Percentage of power savings for the proposed encoding techniques.

	Original Power of Bus (mW)	Encoder Power (mW)	Reduced Power of Bus (After Encoding) (mW)	Power Savings
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 has the lowest overhead. It only consists of four levels of logic. In contrast, 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. Table 6 shows the number of gates and the area required for the proposed encoders.

Table 6 - Comparison of the encoder hardware for the proposed techniques.

	Number of Gates	Area ($\times 1000\ \lambda^2$)
DS	505	488.7
FTS	256	205.8
FMS	313	282.7

6 CONCLUSION

In this paper, we proposed a new approach for bus encoding through sectorization of source word space. The sectorization can be either dynamic or fixed. Adaptive sectorization is a more general approach, whereas the fixed sectorization requires less computational effort. Both techniques can be extended to include as many sectors as desired. These techniques can effectively decrease the transition count of streams with sectorized behavior. Thus, they are very good candidates for encoding over shared

buses where an uncorrelated address is dispatched by my multiple sources. We compared different approaches in terms of power, delay and scalability. For the multiple fixed-sector method, we identified a technique that partitions the sectors evenly. We also showed that using our methods, up to 52% power reduction can be achieved for an external data address bus and 42% reduction for a multiplexed bus between internal cache and external memory.

7 REFERENCES

[1] D. Patterson, J. Hennessy, *Computer Architecture, A Quantitative Approach*, 2nd. ed., Morgan Kaufmann Publishers, 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," *Proc. of 7th Great Lakes Symposium on VLSI*, 1997, pp. 77-82.

[3] W. Fornaciari, M. Polentarutti, D.Sciuto, and C. Silvano, "Power Optimization of System-Level Address Buses Based on Software Profiling," *Proc. Intl. Symposium of Hardware/Software Codesign*, 2000, pp. 29-33.

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

[5] P. Panda, N. Dutt, "Reducing Address Bus Transitions for Low Power Memory Mapping," *Proc. Intl. Symposium of Design Automation and Test in Europe*, 1996, pp. 63-67.

[6] E. Musoll, T. Lang, and J. Cortadella, "Exploiting the Locality of Memory References to Reduce the Address Bus Energy," *Proc. Intl. Symposium on Low Power Electronics and Design*, 1997, pp. 202-207.

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

[8] M. Mamidipaka, D. Hirschberg, N. Dutt, "Low Power Address Encoding Using Self-Organizing Lists," *Proc. Intl. Symposium on Low Power Electronics and Design*, 2001, pp. 188-193.

[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*, Vol. 7, No. 2, pp. 212-221, Jun. 1999.

[10] Y. Aghaghiri, F. Fallah, M. Pedram, "Irredundant Address Bus Encoding for Low Power," *Proc. Intl. Symposium on Low Power Electronics and Design*, 2001, pp. 182-187.

[11] L. Macchiarulo, E. Macii, M. Poncino, "Low-energy for Deep-submicron Address Buses," *Proc. Intl. Symposium on Low Power Electronics and Design*, 2001, pp176-181.

[12] www.simplescalar.org

[13] www.spec.org

[14] www.arm.com/support/amba

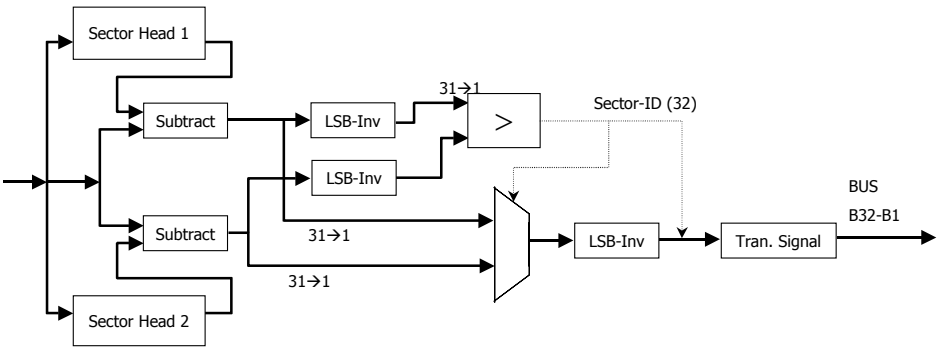

Figure 4 – The DS Encoder.

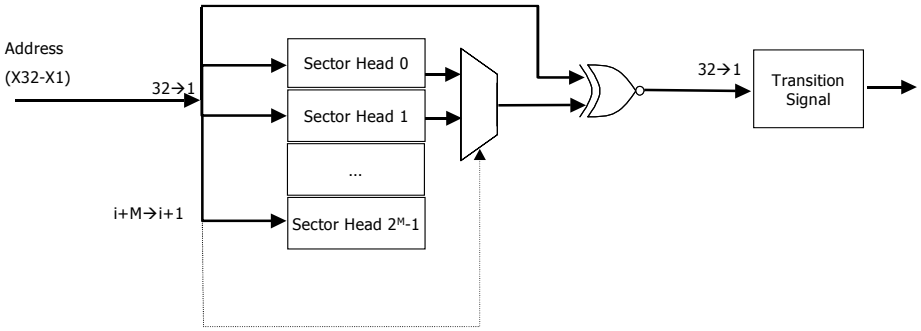

Figure 5- The FMS Encoder.