FV Encoding for Low-Power Data I/O *

Jun Yang Rajiv Gupta

Department of Computer Science
The University of Arizona
Tucson, Arizona 85721

Abstract

Since the energy consumed by I/O pins of a CPU is a significant source of energy consumption, work has been done on developing encoding schemes for reducing switching activity on external buses. Modest reductions in switching can be achieved for data and address busses using a number of general encoding schemes. However, by exploiting the characteristic of memory reference locality, switching activity at address bus can be reduced by as much as 66% [3]. Till now no characteristic has been identified that can be used to achieve similar reductions in switching activity on the data bus. We have discovered a characteristic of values transmitted over the data bus according to which a small number of distinct values, called **frequent values**, account for 58-68% of transmissions over the external data bus. Exploiting this characteritic we have developed an encoding scheme that we call the **FV** (frequent value) **encoding** scheme. To implement this scheme we have also developed a technique for dynamically identifying the frequent values. Our experiments show that FV encoding of 32 frequent values yields an average reduction of 42.7% (with on-chip data cache) and 67.63% (without on-chip data cache) in data bus switching activity for SPEC95 benchmarks. Moreover the reduction in switching achieved by FV encoding is over **four** times the reduction achieved by the **bus-invert** [4] coding scheme and around **six** times the reduction achieved by the **adaptive method** in [6].

Keywords - low power data buses, I/O pin capacitance, internal capacitance, switching, encoding.

1 Introduction

In CMOS circuits most power is dissipated as dynamic power for charging and discharging of internal node capacitances. Thus, researchers have investigated techniques for minimizing the number of transitions inside the circuits. The capacitances at I/O pins are orders of magnitude higher than internal capacitances. Thus, the power dissipated by an IC at these I/O pins is even greater than that dissipated at internal capacitances. Therefore techniques for minimizing switching at external address and data buses, even at the expense of a slight increase in switching at internal capacitances, are being investigated for reducing power consumption [1, 2, 3, 4, 5, 6, 7, 8, 9]

Many of the encoding schemes, such as the bus-invert coding [4], are general and can be applied to both address and data buses. General techniques can only provide modest reductions in switching activity. This is because the characteristics of values sent over data and address buses vary and thus using the same technique for both types of buses cannot be the most effective solution. To obtain greater reductions we must identify special characteristics for information transmitted over address and data buses. Using such a specialized approach significant success has resulted from research into minimizing switching at external address buses. In particular, the technique described in [3] is particularly effective as it reduces the address bus actvity by as much as 66%. The key to achieving such high reductions by this technique is its ability to exploit memory reference locailty. Instead of transmitting a sequence of complete addresses that exhibit locality, in this technique, the offset of current reference with respect to the previous reference to the same *working zone* is sent over the bus, along with an identifier of that zone. Since the offsets are quite small, in comparison to complete addresses, one-hot encoding can be used to transmit them and thus the number of transitions is greatly reduced.

The goal of this work was to develop a technique for data buses that is as effective as the technique in [3] for address buses. Till now an effective specialized approach for a CPU's external data buses has illuded researchers. This is because no suitable characteristic for values transmitted over a data bus has been found. Unlike memory references that exhibit locality, the data values do not exhibit similar locality. In fact the values transmitted over data bus may vary widely accross the range of representable values. We have recently discovered a characteristic of data values sent over a data bus that can be employed to develop an effective encoding scheme. Recently we have shown that a small number of distinct values, frequent values, occupy majority of the data locations in memory for a wide range of application programs [10]. Thus, these values are transmitted very frequently over the data bus.

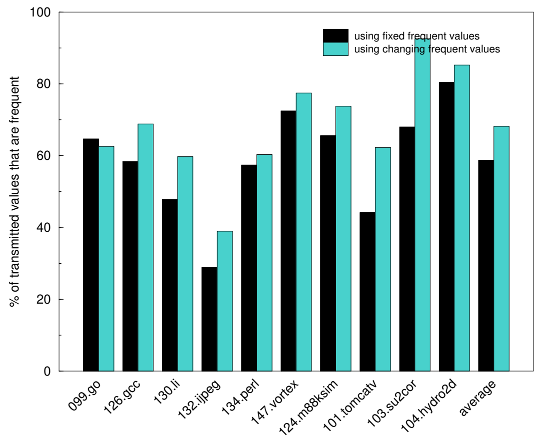


Figure 1: Data bus traffic due to frequent values.

In Figure 1 we show the percentage of total data bus traffic that is the result of transferring frequent values for

*Supported by DARPA award no. F29601-00-1-0183 and National Science Foundation grants CCR-0096122, EIA-9806525, and CCR-9996362 to the University of Arizona.

SPEC95 benchmarks. The two sets of data correspond to situations where the set of frequent values, 32 in all, was kept *fixed* and allowed to *change* over the lifetime of the program. As we can see, on an average 58% (68%) of the traffic is created by fixed (changing) frequent values.

The remainder of the paper is organized as follows. In section 2 we present the FV encoding scheme in detail and also describe how the frequent values are identified by our scheme. In section 3 we describe the result of experiments. In section 4 we discuss related work and experimentally compare our method with existing techniques. Conclusions are given in section 5.

2 Frequent Value Encoding

Now we present the design of our encoder and decoder used to reduce the switching activity on the data bus. Our overall approach is as follows. The frequent values are transmitted over the bus in encoded form while the infrequent values are transmitted in their original nonencoded form. The set of frequent values are kept in a table implemented as a CAM at both the encoder and decoder. This table is searched and if the value to be transmitted is found in it, then the value is regarded as a frequent value which is then transmitted in encoded form. In order to ensure that the decoder can determine whether the transmitted value is in encoded form or not, additional *control* signal must be sent from the encoder to the decoder in some situations. As we describe later in this section, our method for maintaining frequent values is such that the contents of the frequent value tables at both the encoder and decoder are identical to each other.

In the remainder of the section we first describe our encoding scheme in detail. Then we describe our method for finding frequent values, that is, we describe how the contents of the frequent value tables are maintained.

2.1 Encoding-Decoding Scheme

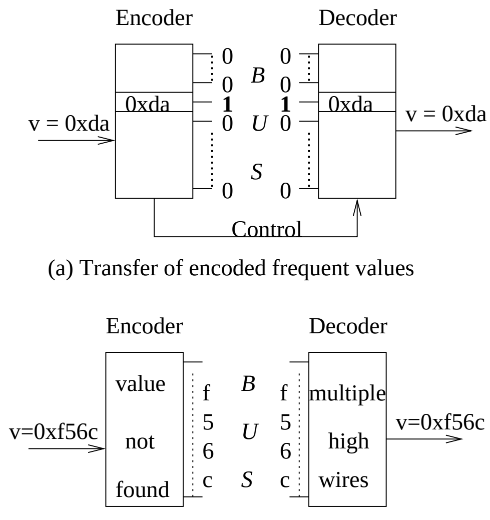
The base FV Encoding scheme. Our method for encoding frequent values has the flavor of one-hot encoding with one important difference. Our encoding scheme overcomes the major drawback of one-hot encoding in that it does not need exponential number of wires (2^n, where n is the number of bits representing the value) to transfer data. Instead, it uses the same number of wires as the data bus width (we assume that this number is 32) and maintains the desired property of low switching activity.

We are able to achieve the above goal as follows. The "hot" wire generated from the encoder is not used to represent the true decimal value being transfered but rather it indicates in which entry of the frequent value table in the encoder or decoder that frequent value can be found. In other words if the i^{th} entry is found to hold the same value as the one to be transmitted, the i^{th} output wire is set as 1 and all the remaining wires are set as 0. Thus a **code** with only one wire at high voltage is formed and sent over the data bus, completing the coding process.

When the decoder receives the code from the bus, it reads out the value from the i^{th} entry indicated by the code. We will show later how our method for maintaining the contents of the tables at the encoder and decoder ensures that the contents of the two tables are identical and thus the value is correctly decoded.

The infrequent values are transmitted in unencoded form. If a value to be transmitted is an infrequent value it cannot be found in the encoder CAM. Thus, the encoder does not attempt to generate a code. Instead, it simply passes the original value onto the data bus. When the decoder receives the value and finds more than one hot wires in it, it concludes that the transmitted value is not encoded.

It is possible that a value being transmitted in unencoded form contains a single high bit and all of its remaining bits are zeros. We ensure that the decoder does not incorrectly decode this value by sending a single bit control signal from the encoder telling the decoder to skip decoding. Our experimental results also include the switching overhead from sending the control signal.



(a) Transfer of encoded frequent values

(b) Transfer of unencoded infrequent values

Figure 2: Encoding-Decoding Setup.

An example. Figure 3 illustrates how the FV encoding scheme is able to reduce the amount of switching. It compares the switching activity on a sample sequence of values with and without the use of the FV scheme. Here we assume the initial value on the data bus is 0 which is followed by three frequent values and one infrequent value shown in the column *binary value* of Figure 3. The values in this example were taken from a run of the 126.gcc program. In the *no encoding* case the values are transmitted as is while in the FV encoding case the coded values that are transmitted are shown in Figure 3. The number of bit switches are shown for both the cases. As we can see, the peak reduction in switching can be as high as 30 ($= 32 - 2$) and in general the reduction is also high.

This example also illustrates that not only do we need to encounter frequent values but also for good performance they must appear back to back. A pattern of intervening frequent and infrequent values is not favorable to our scheme. We performed an experiment to see how often a frequent value is preceded by a frequent value. Figure 4 plots the percentage of total transmitted values that are frequent and preceded by a frequent value. We found that in SPEC95 suite this number is quite significant. On an average we see that 45% and 60% of all values belong to the above category for fixed and changing set of frequent values respectively. These statistics suggest that our encoding scheme should reduce switching activity.

Binary Value	No Encoding	FV Encoding	
	Switching	Transmitted Value	Switching
01000000001101010001001011010000	10	$0_{0..5}10_{7..31}$	10
00000000000000000000000000100000	11	$0_{0..21}10_{23..31}$	2
11111111111111111111111111111111	31	$0_{0..6}10_{8..31}$	2
00000000000000000000000000000101	30	$0_{0..28}101$	3

Figure 3: Illustrating reduction in switching transitions using FV encoding.

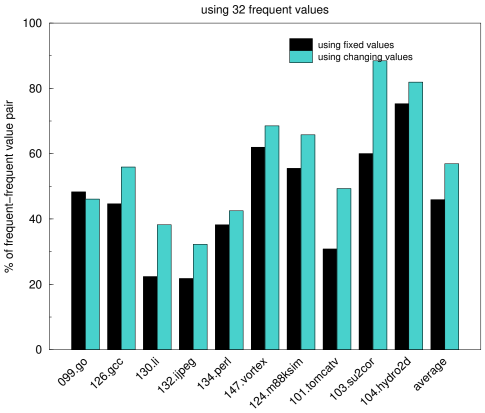


Figure 4: Occurrence of frequent values in sequence.

Further reduction in switching. With continuously occurring frequent values, FV encoding will send out codes one after another with "hot" bits at different position, resulting in 2 switching transitions for each pair of frequent values (see Figure 3). We can further reduce the switching down to 1 using an approach used in [3, 6]. By XORing current code with the previous value sent over the data bus as shown in Figure 5 such a reduction is achieved. In Figure 3, as an example, the *transmitted value* column, starting from the 2*nd* row, becomes $0_{0..5}10_{7..21}10_{23..31}$, $0_{0..5}110_{8..21}10_{23..31}$ and $0_{0..5}110_{8..21}10_{23..28}101$. The *switching* column for FV encoding becomes 1, 1 and 2.

$$Send_n = Send_{n-1} \oplus Code_n$$
$$DCode_n = Send_n \oplus Send_{n-1}$$

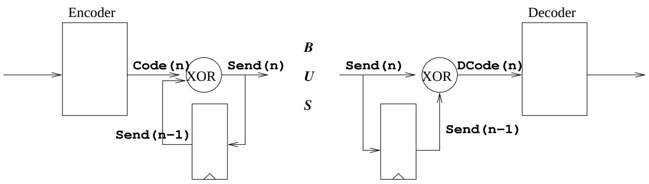


Figure 5: Reducing switching between encoded values.

2.2 Identifying Frequent Values

Having described our encoding scheme, let's now discuss how we fill and update the encoder and decoder tables with data values. There are two ways of doing that we consider in this paper:

1. A *fixed set* of values known in advance to initialize both encoder and decoder can be used. The set of values can be obtained through ranking of the frequency of values that appeared in a previous run of the program.

2. A *changing set* of frequent values can be maintained as the program runs. Thus, the contents of the frequent value tables adapt to changes in the frequent values for different parts of execution.

Using fixed values to preset the encoder and decoder has the advantage that the coders do not have to change the table contents dynamically thus reducing the runtime overhead. However, it requires that values be known before hand and different program needs different values. The second method, on the other hand, does not need a priori information of data values and does not distinguish among different programs. With these features, we pay the price of identifying the frequent values on the fly.

In our previous work [10] we have always used a small number of fixed frequent values for encoding. This was because the application we considered in the past was that of storing compressed (encoded) data in data cache to improve on-chip cache performance. Frequent values cannot be varied in this case because every time the encoding changes, the contents of the cache must be flushed or reencoded. This operation is too costly and therefore not practical. However, the application that we are considering in this paper does not have similar constraints – we can easily exploit changing frequent values by changing the contents of the frequent value tables.

The changing frequent value scheme has the potential for giving better performance. This is beacuse a value with high frequency in one span of time may not occur as frequently in another span of time during a program run. We conducted an experiment to determine whether the need for an adaptive scheme exists. In this experiment we divided the execution of a program into smaller time intervals and for each of these intervals we found the best 32 frequent values. We considered the commonality between this nearly ideal set of values and the values used in our changing value scheme (described later in this section) as well as the fixed set of frequent values. The plot in Figure 6 shows that the overlap between the changing set and ideal set is much greater (around 20 or higher for most of the time) than the overlap between the fixed set and the ideal set (slightly less than 10). This plot is for the `su2cor` benchmark and represents a time period which corresponds to 25% of the program run over which three million values were transmitted over the data bus. We favor the dynamic encoding scheme but will also include experimental results for fixed value scheme in the experimental section. Next we will illustrate how we find the frequent values dynamically.

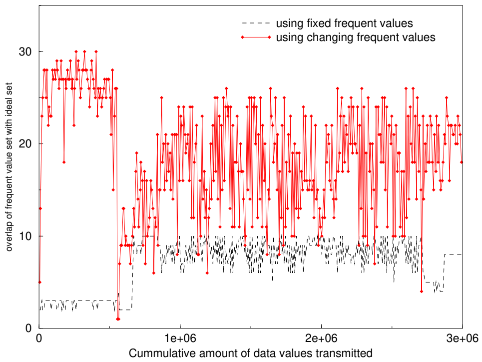


Figure 6: Changing set vs fixed set.

LRU policy. We use the LRU replacement policy for filling and updating both encoder and the decoder frequent value tables. To gain time ordering information, we use a *reference bit* and a n-bit timestamp for each value recorded in the coder. The reference bit is set when the value appears at the input. At regular intervals, the reference bit is shifted right into the high-order bit position of the n-bit timestamp causing all bits in the timestamp also to be shifted right and the lowest-order bit in the timestamo being discarded. This operation is performed for all entries in the two tables and at the same time all the reference bits are reset. Thus, the timestamp keeps the history of value occurences for the last n time periods. The timestamp of 000 means this value did not appear during the last three time intervals, timestamp 100 means it was just seen in the last interval, and the timestamp 000 with reference bit set means it is encountered in the current time slot. When an entry is required and a value is to be evicted, the entry that is selected is the one with the smallest timestamp and clear reference bit. The new value is put in with a fresh reference bit and timestamp (all 0's) in this selected entry.

Figure 7 illustrates the above process using a sequence of data values. At $t8$, the contents of the frequent value table along with reference bit and timestamp are shown in (a). Suppose at this time we need to update the timestamp, (b) shows all the changes made to the timestamp and the reference bit. At $t9$, a new value replaces the value -1 because it has the smallest timestamp 000 (c). New value also gets a fresh timestamp and a 1 as reference bit. At $t10$, another new value replaces the value $0xae2$ because it has the smallest timestamp and its reference bit is 0 (d). The values in the frequent value table, together with the timestamp, gives an idea on what are the values most recently occured and therefore might be seen again soon.

Keeping encoder/decoder consistent. It is extremely important to keep the sender side encoder and the receiver side decoder consistent all the time. We use the **same** replacement policy for both to assure they contain the same values. In more detail, if there are multiple entries that have the same timestamp, both the encoder and the decoder follow the same rule for picking up a victim, say the first victim they encounter during searching. By doing so, we guarantee both sides contain not only the same values but also the same indices for every value. The grounds for this to be true is that they have the same timestamp value and reference bit. This can be easily achieved by using the same time interval for updating the timestamp and the reference bit.

Time:	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10
Value:	0	−1	1	0xae2	0	0	0	1	0x40457f80	7

FV table	ref.	ts
0	1	101
−1	0	001
1	1	010
0xae2	0	010

(a)

FV table	ref.	ts
0	0	110
−1	0	000
1	0	101
0xae2	0	001

(b)

FV table	ref.	ts
0	0	110
0x40457f80	1	000
1	0	101
0xae2	0	001

(c)

FV table	ref.	ts
0	0	110
0x40457f80	1	000
1	0	101
7	1	000

(d)

(a) The contents of FV table, reference bit(ref) and timestamp(ts) at t8

(b)Updating ref. and ts: Shift ref. right to the highest order bit of ts, shifting rest of ts right by one bit

(c) At t9, value −1 is replaced by new value 0x40457f80

(d) At t10, value 0xae2 is replaced by new value 7

Figure 7: Example of frequent value identification.

3 Experimental Evaluation

We conducted experiments by executing a subset of the `SPEC95` benchamrks and measuring the switching activity on the external data bus. The results of the various experiments conducted are described next.

Switching activity reduction. We measured the reduction in switching activity for the two schemes: one in which the frequent values are kept fixed and the other in which they are allowed to dynamically change during program execution. These results were gathered for varying number of frequent values (2, 4, 8, 16 and 32). The results are given in Figure 8. From the data in Figure 8 we can draw three conclusions.

- The reduction in switching activity at the data bus is greatly reduced using FV encoding when the set of frequent values is allowed to change. An average reduction of 42.7% and a maximum reduction of 81.65% is observed when a changing set of 32 frequent values are used.

- We can see that the reductions in switching activity increases significantly with an increase in allowable frequent values. The average reduction increases from 15.2% for 2 changing frequent values to 42.7% for 32 changing frequent values.

- Finally the changing value scheme outperforms the fixed frequent value scheme by greater and greater degree as the number of allowable frequent values is increased. For 32 values the fixed value scheme provides only 29.06% reduction in contrast to 42.7% reduction in switching achieved by changing value scheme.

Impact of timestamp size. We also studied the performance of changing value FV encoding method by varying the size of timestamp used to carry out replacement of frequent values. The results in Figure 9 show that increasing the size of timestamp has very little impact on performance. Therefore using a one bit timestamp is sufficient in practice.

Program	2 freq. values		4 freq. values		8 freq. values		16 freq. values		32 freq. values	
	fixed	changing	fixed	changing	fixed	changing	fixed	changing	fixed	changing
099.go	24.12	18.49	25.23	20.67	27.45	25.61	34.81	33.15	38.02	41.51
126.gcc	13.88	16.53	14.26	18.54	14.71	23.12	18.60	31.29	24.14	39.35
130.li	10.09	11.46	10.29	13.71	12.82	18.64	17.26	27.85	24.40	40.58
132.ijpeg	-10.71	-8.46	-10.61	-4.93	-9.48	1.05	-8.12	5.16	-5.78	9.56
134.perl	14.13	16.55	23.17	18.75	26.39	23.17	30.55	30.70	38.32	42.54
147.vortex	8.36	12.05	11.04	14.71	14.16	19.19	17.23	24.94	21.90	31.17
124.m88ksim	22.04	25.57	27.19	30.47	29.00	35.29	32.23	42.67	37.93	49.03
101.tomcatv	4.30	4.33	9.51	9.13	12.06	17.38	15.82	28.37	20.57	43.57
103.su2cor	30.51	38.55	30.80	48.13	39.39	60.86	41.91	72.58	52.06	81.65
104.hydro2d	0.18	16.97	19.02	31.93	33.62	41.04	36.49	46.28	39.09	48.05
Average	11.69	15.20	15.99	20.11	20.01	26.54	23.68	34.30	29.06	42.70

Figure 8: Switching activity reduction in data bus.

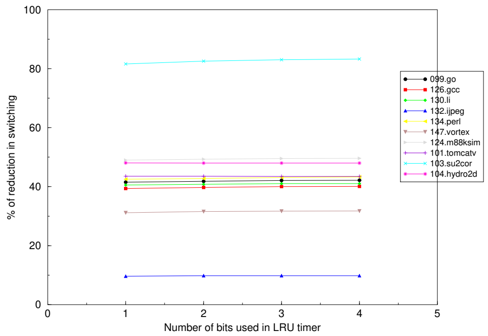


Figure 9: Timestamp size vs switching reduction.

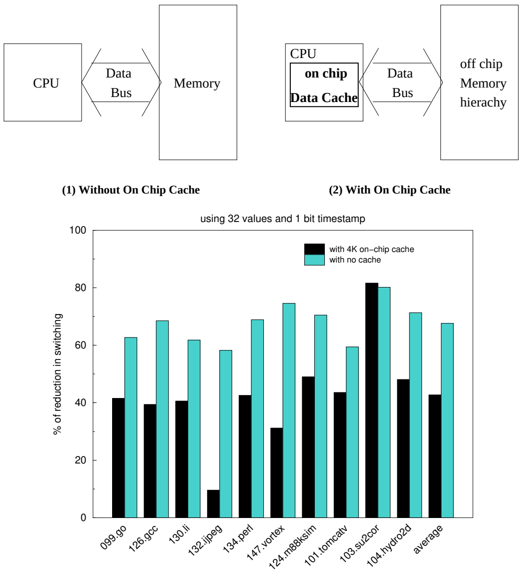


Figure 10: With on-chip cache vs without on-chip cache.

Performance without on-chip cache. In all the experiments described so far we assumed that there was an on-chip data cache present. We also repeated our experiments without an on-chip cache. This is because in many embedded and DSP processors from AT&T Microelectronics, Motorola, Zilog and Texas Instruments there is no on-chip cache. The results in Figure 10 show, in the absence of an on-chip cache the reductions in switching activity are even greater. The average reduction for 32 changing values increases from 42.7% to 67.63%.

4 Related Work

There has been abundant research done in reducing address bus switching, based on the sequentiality of program counters [1, 2, 8] and regularity of memory accesses [3]. The work that applies to data buses falls in two categories: (a) general techniques that apply to both data and address buses; and (b) techniques specifically developed for data buses. Now we compare our technique with techniques in each of these categories.

Comparison with the general bus-invert coding scheme.

A well known general technique for reducing switching is the bus-invert coding scheme. In this scheme the Hamming distance between the present bus value and the next value is computed. If this is greater than half the number of total bits, then the data value is transmitted in inverted form. An additional bit, the invert signal, is also sent to indicate how the data is to be interpreted at the other end. We implemented this technique to compare its performance with FV encoding for data buses. The results in Figure 11 show that on an average bus-invert scheme reduces switching by 10.44%. In contrast the FV encoding scheme with 32 changing (fixed) values reduces switching by 42.7% (29.06%). Thus, the changing (fixed) value FV encoding scheme outperforms the bus-invert coding method by nearly a factor of 4 (2).

Comparison with other data bus techniques.

Some of the work in this category [5, 6] starts from statistical properties of the data streams and compute codes such that value pairs with higher probablity of occurrence lead to fewer switching transitions. In [5], the authors introduce a generic encoder-decoder architecture model and provide a few sample solutions in each module of the generic model. In [6], the

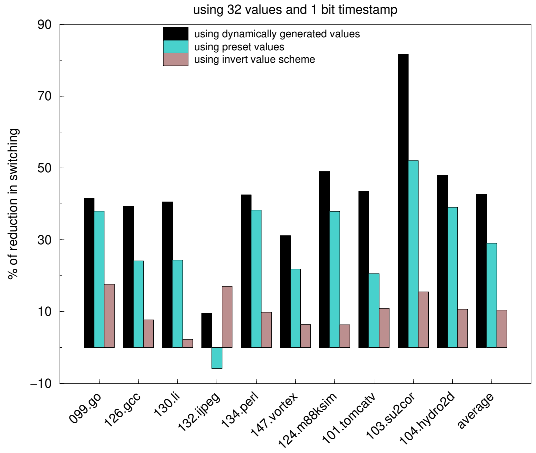


Figure 11: Bus-invert vs FV Encoding.

authors introduce two heuristic approximations to the theoretical algorithm whose performance is less than satisfactory. Both papers first emphasize algorithms with prior knowledge of statistics on input data streams and then they both provide adaptive methods to remove this constraint. However, either the adaptive method requires expensive hardware or it does not perform well. Figure 12 compares our FV encoding (32 dynamically changing set of values) method with the adaptive method in [6]. As we can see, on an average, the reduction in switching using FV encoding is nearly six times of that achieved using the adaptive method.

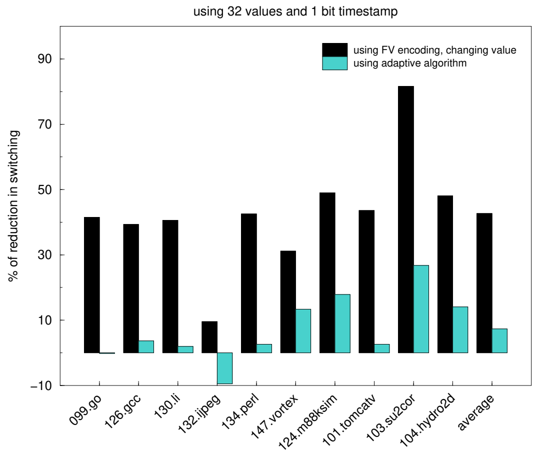


Figure 12: FV vs. Adaptive Encoding in [6]

Thus, the above experiments show that our lightweight FV encoding scheme is quite effective in comparison with other proposed techniques for use in CPU data buses.

5 Conclusions

In this paper we have demonstrated that by exploiting the characteristic of frequently transmitted values, we can design the FV encoding scheme which reduces the switching activity on an external data bus substantially. The reductions are even greater for processors without on-chip caches. Furthermore we have demonstrated that the frequent values at any point during execution can be effectively identified using a simple hardware mechanism. By allowing the set of frequent values to change during execution we obtain reductions in switching that are substantially greater than reductions achieved by a scheme that uses fixed set of frequent values for the entire execution. Finally we have demostrated that FV encoding outperforms both bus-invert coding [4] and the adaptive scheme of [6].

References

[1] 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*, pages 24-30, 1994.

[2] L. Benini, G. DeMicheli, E. Macii, D. Sciuto, C. Silvano, "Asymptotic Zero-Transition Activity Encoding for Address Busses in Low-Power Microprocessor-Based Systems," *Great Lakes Symposium on VLSI*, pages 77-82, Urbana IL, March 1997.

[3] E. Musoll, T. lang, and J. Cortadella, "Exploiting locality of memory references to reduce the address bus energy," *ACM/IEEE International Symposium on Low Power Electronics Design*, pages 202-207, Monterey, CA, August 1997.

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

[5] S. Ramprasad, N. R. Shanbhag, I. N. Hajj, "A Coding Framework For Low-Power Address and Data Busses," *IEEE Transactions on VLSI Systems*, Vol. 7, No. 2, pages 212-221, June 1999.

[6] L. Benini, A. Macii, E. Macii, M. Poncino, and R. Scarsi, "Synthesis of low-overhead interfaces for power-efficient communication over wide buses," *ACM Design Automation Conference*, New Orleans, Louisiana, 1999.

[7] N. Chang, K. Kim, and J. Cho, "Bus encoding for low-power high-performance memory systems," *ACM Design Automation Conference*, pages 800-805, Los Angeles, CA, 2000.

[8] W-C. Cheng and M. Pedram, "Power-optimal encoding for DRAM address bus," *ACM/IEEE International Symposium on Low Power Electronics Design*, pages 250-252, 2000.

[9] M.R. Stan and W.P. Burleson, "Coding a terminated bus for low power," *Great Lakes Symposium on VLSI*, pages 70-73, Buffalo, NY, March 1995.

[10] Y.Zhang, J.Yang, and R.Gupta, "Frequent Value Locality and Value - Centric Data Cache Design, " *The Ninth International Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS), pages 150-159, Cambridge, MA, November 2000.