Frequent Value Encoding for Low Power Data Buses

JUN YANG
University of California at Riverside
RAJIV GUPTA
The University of Arizona
and
CHUANJUN ZHANG
University of California at Riverside

Since the I/O pins of a CPU are 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 buses using a number of general purpose encoding schemes. However, by exploiting the characteristic of memory reference locality, switching activity on the address bus can be reduced by as much as 66%. 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 32% of transmissions over the external data bus. Exploiting this characteristic we have developed an encoding scheme that we call the *FV encoding* scheme. To implement this scheme we have also developed a technique for dynamically identifying the frequent values which compares quite favorably with an optimal offline algorithm. Our experiments show that FV encoding of 32 frequent values yields an average reduction of 30% (with on-chip data cache) and 49% (without on-chip data cache) in data bus switching activity for `SPEC95` and `mediabench` programs. Moreover the reduction in switching achieved by FV encoding is *2 to 4* times the reduction achieved by the *bus-invert* coding scheme and *1.5 to 3* times the reduction achieved by the *adaptive method*. The overall energy savings on data bus we attained considering the coder overhead is 29%.

Categories and Subject Descriptors: B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Input/Output Circuits*; B.4.m [**Input/Output and Data Communications**]: Miscellaneous

General Terms: Algorithms, Design, Measurement

A preliminary version of this article appeared as a brief poster in ISLPED'01.
This work was supported by DARPA award no. F29601-00-1-0183 and National Science Foundation grants CCR-0208756, CCR-0220334, CCR-0105355 and EIA-00801213 to the University of Arizona.
Authors' addresses: J. Yang and C. Zhang, University of California, Riverside, SURGE Bldg., Riverside, CA 92521; email: {junyang,chzhzang}@cs.ucr.edu; R. Gupta, The University of Arizona, Gould-Simpson Bldg., Room 746, 1040 E. Fourth St., Tucson, AZ 85721; email: gupta@cs.arizona.edu.

Additional Key Words and Phrases: 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 at the I/O pins is even greater than that dissipated at internal capacitances. Therefore, techniques for minimizing switching at external address and data buses, at the expense of a slight increase in switching at internal capacitances, have been investigated for reducing power consumption [Benini et al. 1997, 1999; Chang et al. 2000; Cheng and Pedram 2000; Komatsu et al. 2000; Kretzschmar et al. 2001; Ramprasad et al. 1999; Musoll et al. 1997; Stan and Burleson 1995a; Su et al. 1994].

Many of the encoding schemes, such as the bus-invert coding [Stan and Burleson 1995a], are general purpose and can be applied to both address and data buses. General purpose 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 is not the most effective solution. To obtain greater reductions we must identify special characteristics of the 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 Musoll et al. [1997] is particularly effective as it reduces the address bus activity by as much as 66% for some benchmarks. The key to achieving such high reductions by this technique is its ability to exploit memory reference locality. The memory regions being referenced by a program are divided into working zones. 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 switching transitions is greatly reduced.

The goal of this work was to develop a technique for data buses that is similarly effective as the above technique is for address buses [Musoll et al. 1997]. The above technique is effective because it exploits the characteristic of data reference locality. Till now an effective specialized approach for a CPU's external data bus has been illusive. 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 the data bus may vary widely across 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,

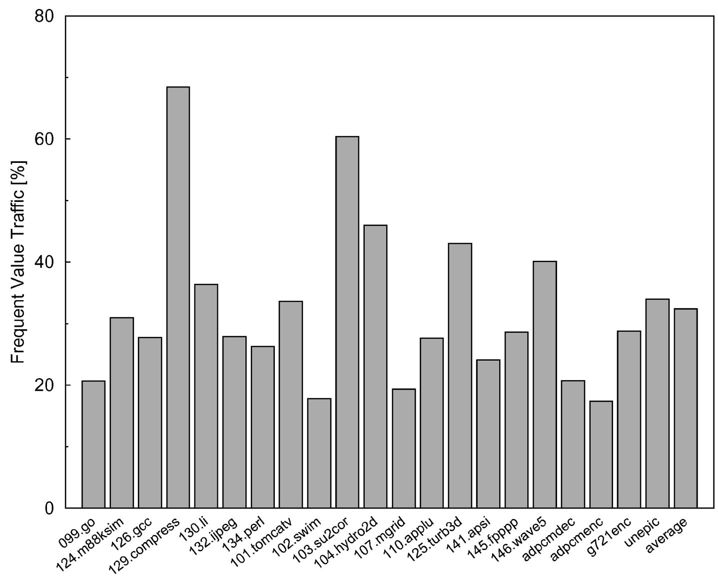


Fig. 1. Data bus traffic due to 32 frequent values.

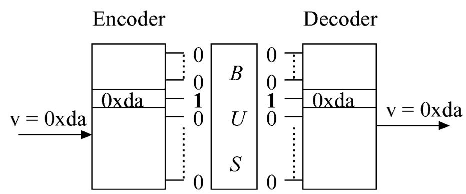
frequent values, occupy majority of the data locations in memory for a wide range of application programs [Zhang et al. 2000; Yang and Gupta 2002]. Thus, these values are transmitted very frequently over the data bus.

In Figure 1, we show the percentage of total data bus traffic that is the result of transferring top 32 frequent values for `SPEC95` and `mediabench` programs. The statistics are obtained by measuring the switching activity on the data bus connecting the CPU and the off-chip memory. Furthermore it is assumed that there are on-chip instruction and data caches each of size 8K bytes. The data transfered over the data bus is a mixture of instructions and program data. The frequent values observed over the data bus are dynamically identified using our proposed algorithm. On average, over 32% of values transmitted are frequent values and this number reaches 68% for `compress`.

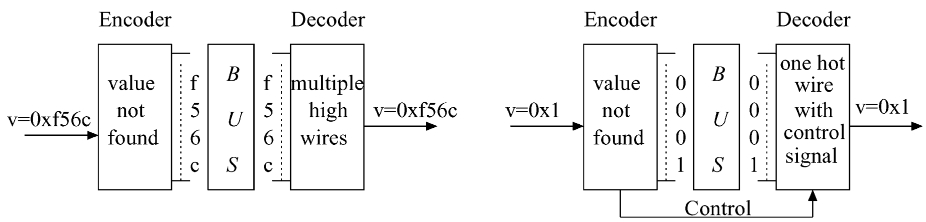
The remainder of the article is organized as follows: In Section 2, we present the FV encoding scheme in detail. In Section 3, we describe how the frequent values are identified by our scheme. In Section 4, we experimentally evaluate the effectiveness of FV encoding in reducing switching activity. In Section 5, we present the detailed design of the FV coder and the overall energy savings. In Section 6, we discuss related work and experimentally compare our method with existing techniques. Conclusions are given in Section 7.

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 nonfrequent values are transmitted in their original unencoded form. The set of frequent values are kept in a table implemented as a content addressable memory (CAM) by



(a) Transfer of encoded frequent values



(b) Transfer of unencoded nonfrequent values

Fig. 2. Encoding-decoding setup.

both the encoder and the 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 the decoder are always identical. In the remainder of the section, we first describe our base encoding scheme in detail and then we describe some enhancements to this base scheme.

2.1 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 require 2^n wires, where n is the number of bits representing the value, to transfer the data. Instead, it achieves low switching activity by using the same number of wires as the data bus width. In our experiments we assume that this number is 32.

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 the frequent value can be found. In other words, if the ith entry in the frequent value table is found to contain the same value as the one being transmitted, then the ith output wire is set to 1 and all the remaining wires are set as 0. This is how a *one-hot code* is formed and sent over the data bus, completing the coding process (see Figure 2(a)). When the decoder receives the code from the bus, it reads out the value from the ith entry indicated by the

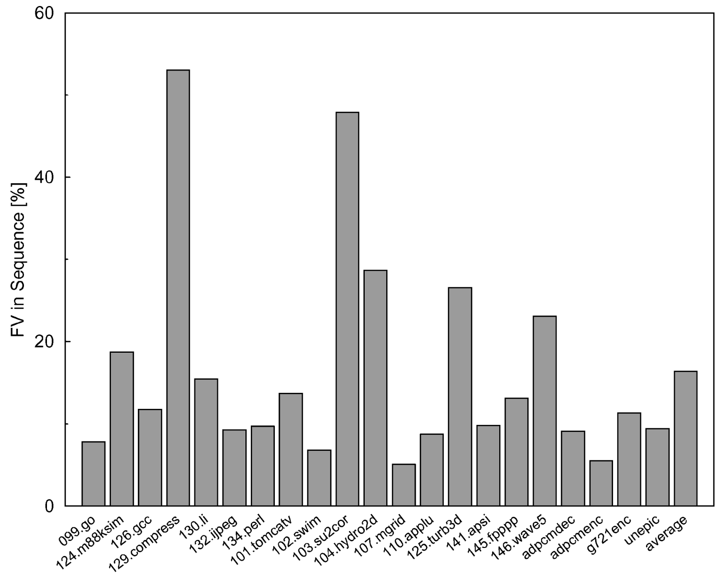


Fig. 3. Occurrence of frequent values in sequence.

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. Under the above scheme, if frequent values are transmitted back to back, then at most two bits switch while all other bits remain zero. This is how FV encoding reduces switching activity.

The nonfrequent values are transmitted in unencoded form. If a value to be transmitted is a nonfrequent 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 (see Figure 2(b)).

It is possible that a nonfrequent 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 erroneously decode this value by sending a single bit control signal from the encoder telling the decoder to skip decoding (see Figure 2(c)). Our experimental results also include the switching overhead from sending the control signal.

2.2 Enhancements of Base FV Encoding Scheme

2.2.1 *XORing Values.* The base encoding scheme reduces switching to at most 2 bits if a frequent value being transmitted is also preceded by a frequent value. While our base encoding scheme gives good performance when frequent values are encountered back to back, a pattern of intervening frequent and nonfrequent values is not favorable to our base scheme. In Figure 3, the percentage of traffic due to frequent values that are also preceded by frequent value

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

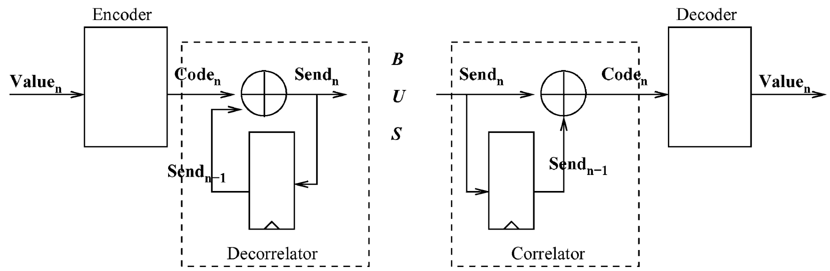


Fig. 4. Reducing switching by XORing values.

transmissions is given. On an average, this number is 16%. From the data presented earlier in Figure 1, we know that on an average the frequent values account for 32% of the overall traffic. Therefore, on an average 16% of transmitted values are frequent values that are preceded by nonfrequent values.

We can also reduce switching between nonfrequent and frequent value transmissions using a decorrelator described in Musoll et al. [1997] and Benini et al. [1999]. If we take the *XOR* of the current value to be transmitted ($Code_n$) and the previously transmitted value ($Send_{n-1}$), then this has the effect of flipping only those wires of the bus that were low when $Send_{n-1}$ was sent and are high in $Code_n$. Therefore, if $Code_n$ corresponds to a frequent value, it contains only one high bit and, therefore, no matter whether it is preceded by a frequent value or a nonfrequent value (i.e., $Send_{n-1}$ is frequent or nonfrequent) the switching activity is only 1 bit. In other words, transmission of a frequent value always results in switching of one bit. The combination of FV encoding and XORing current code with the previous value sent over the data bus is shown in Figure 4.

2.2.2 *Equality Test.* XORing the values can help reduce switching when different codes are to be transmitted in sequence. However, it also brings unnecessary switching when the same code is transferred repeatedly. For example, if a code with the ith bit hot was transferred n times continuously, the switching on bus will toggle n times at the ith wire. This increases switching since transferring same code should not induce any switching while in our case it does cause 1 switch, and eventually it can increase overall switching. Figure 5 shows how often this situation arises. It gives the percentage of traffic due to transmission of a code that is immediately followed by the same code. On an average, this situation accounts for 16% of the traffic. It is observed that this characteristic is observed at low levels in all benchmarks. However, for a few benchmarks (e.g., `compress` and `turb3d`), this situation is very common. As a result, for these benchmarks in particular, we should avoid the switching caused due to repeated transmission of the same code.

The additional switching can be removed easily as shown in Figure 6. We keep a register of the last value ($Value_{n-1}$) transferred and compare it with the current value ($Value_n$). If the two values match, we send the code for the last value ($Code_{n-1}$) on the bus again. The receiving side, without knowing the

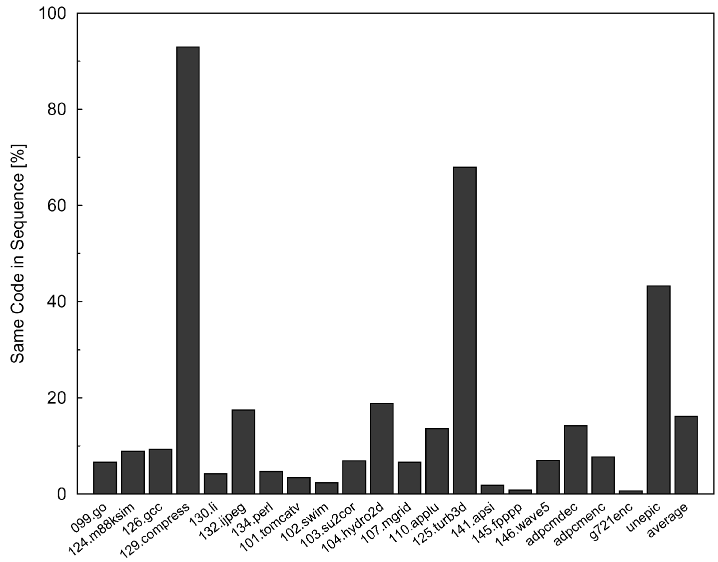


Fig. 5. Transmission of identical code in sequence.

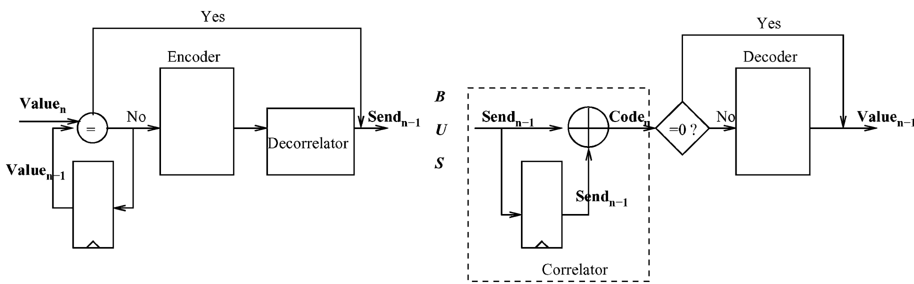


Fig. 6. Dealing with equal code transfer.

equality property of the current value, puts the code through the correlator. Since the code is the same as the last code, the correlator, namely XOR, will compute the result 0 as $Code_n$. There are now two cases where $Code_n$ can be 0: one is when $Send_{n-1}$ is sent twice back to back as we just explained; and the other is when $Value_n$ is 0 and 0 is not a frequent value and therefore not encoded. We can disambiguate the two cases by hardwiring an entry in the encoder/decoder to 0 and thus making 0 a permanent frequent value which is therefore always transmitted in encoded form. This leaves only one possibility for $Code_n$ to be 0, which corresponds to the case when the same value is being transfered again and therefore the decoder can simply output the last value it produced. Note that, in this process, the sending side did not initiate the activity of the encoder or the decorrelator and the receiving side used only the correlator. Thus, the energy spent in the encoder and the decoder is also saved.

2.2.3 *Hamming Distance Based Exclusion of Frequent Values.* So far, in our discussions we have considered all encountered values as candidates for being frequent values. However, it should be noted that not all frequent values are equally effective in reducing switching activity. The impact of encoding a frequent value is proportional to the hamming distance between the unencoded frequent value and the corresponding one-hot code assigned to it. The hamming distance between a small unsigned value and a one-hot code is quite small. Therefore, whether these small unsigned values are transmitted in unencoded form or in a one-hot encoded form, the switching activity that will occur will be very close. It is possible that by excluding such values from consideration during frequent value identification, we may achieve better performance. First their exclusion will allow entries in the frequent value table to be used by other values which are not as frequent but have a greater hamming distance from the one-hot code they are assigned. Second the encoding and decoding activity will be reduced because the frequent value table need not be accessed for these values at all.

2.3 An Example

Figure 7 illustrates how the FV encoding scheme and its enhancements are able to reduce the switching activity. It compares the switching activity for a sample sequence of values without encoding and with different levels of encoding. Here we assume the initial value on the data bus is 0 which is followed by two frequent values, one nonfrequent value, and finally two more frequent values shown in the first column of the first table in Figure 7. All values are written in hexadecimal format. If no encoding is carried out the number of bit transitions for this sequence is 32. This number reduces to 9 when the frequent values are encoded using the base FV encoding scheme. The reductions arise due to transmission of one hot-codes as opposed to original values with large numbers of high bits. The application of XOR reduces bit transitions by one bit during transmission of second, third and fourth values. However, it also increases the bit transitions for the last value from no bits to 1 bit transition. By performing the equality test this additional bit transition can be avoided leading to the final bit transition count of 5 bits.

3. IDENTIFYING FREQUENT VALUES

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

(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 internal switching overhead. However, it requires that values

Transmitted Binary Value Without Encoding	Switching
0xff	8
0xfff	4
0x300	10
0xfff	10
0xfff	0
Total Switching	32

Frequent Values	One-hot Code
0xff	0x1
0xfff	0x2

Transmitted Binary Value With FV Encoding	Switching
0x1	1
0x2	2
0x300	3
0x2	3
0x2	0
Total Switching	9

Transmitted Binary Value With FV Encoding + XOR	Switching
0x1	1
0x3	1
0x303	2
0x301	1
0x303	1
Total Switching	6

Transmitted Binary Value With FV Encoding + XOR + Equal	Switching
0x1	1
0x3	1
0x303	2
0x301	1
0x301	0
Total Switching	5

Fig. 7. Illustrating reduction in switching transitions using FV encoding.

be known beforehand. Since different programs have different frequent values, a profiling run is needed to identify the frequent values. Our prior experience shows that frequent values are relatively insensitive to program input and therefore they can be identified once using a single profiling run and repeatedly used in all future executions [Zhang et al. 2000; Yang and Gupta 2002].

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. The changing frequent value scheme has the potential for giving better performance. This is because a value with high frequency in one span of time may not occur as frequently in another span of time during a program run.

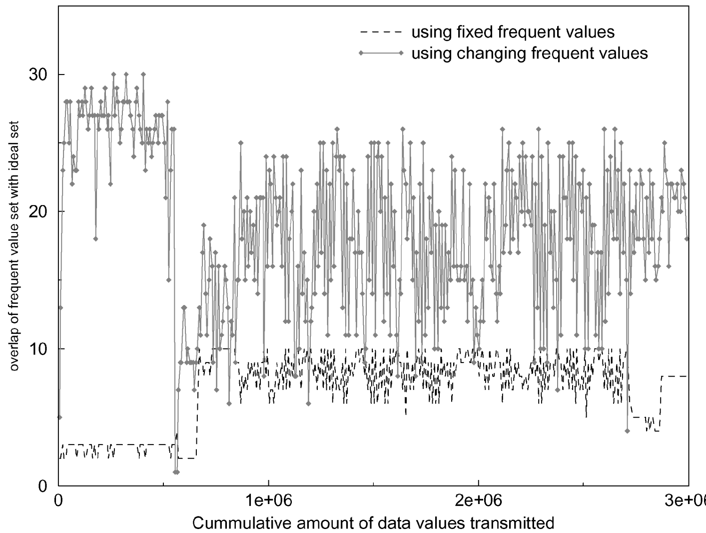


Fig. 8. Changing frequent value set vs. fixed frequent value set.

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 8 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.

3.1 LRU Replacement 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 an 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 timestamp 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 occurrences 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

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

Coder	ref.	ts
0	1	101
−1	0	001
1	1	010
0xae2	0	010

(a)

Coder	ref.	ts
0	0	110
−1	0	000
1	0	101
0xae2	0	001

(b)

Coder	ref.	ts
0	0	110
0x40457f80	1	000
1	0	101
0xae2	0	001

(c)

Coder	ref.	ts
0	0	110
0x40457f80	1	000
1	0	101
7	1	000

(d)

(a)The contents of coder, 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

Fig. 9. Example of frequent value identification.

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 9 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 as shown in (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 as shown in (d). The values in the frequent value table, together with the timestamp, gives an idea on what are the values most recently occurred and therefore might be seen again soon.

3.2 Keeping Encoder and Decoder FV Tables 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 the search. By doing so, we guarantee both sides contain not only the same values but also the same indices for every value. The basis for this to be true is that they have the same timestamp value and reference bit. This is easily achieved by using the same time interval for updating the timestamp and the reference bit.

4. EXPERIMENTAL EVALUATION OF SWITCHING ACTIVITY REDUCTION

We conducted experiments by executing the `SPEC95INT`, `SPEC95FP`, and a subset of `mediabench` programs. The goals of these experiments were as follows:

—Measuring the reductions in *switching activity* on the external data bus due to FV encoding and its enhancements;
—Measuring the degree of on-chip *encoding and decoding activity* for FV encoding and its enhancements;
—Measuring the impact of varying the *number of frequent values* and their *width* on switching reduction;
—Accuracy of our *frequent value replacement* algorithm and its comparison with a *perfect online LRU* replacement and *optimal offline* replacement algorithms;
—The impact of absence of *on-chip caches* on switching activity reduction; and
—Comparison of our technique with *bus-invert* and *adaptive* encoding schemes.

4.1 Switching Activity Reduction

4.1.1 *Effectiveness of FV Encoding and Its Enhancements.* We tested the impact of each component in our FV encoding scheme in reducing switching. The purpose is to answer the question: do we need all the components and if yes how much benefit does each one bring? To see this, we first considered the following three configurations of a frequent value based encoding algorithm:

(1) *FV Encoding Only*. This is the base FV encoding algorithm.
(2) *FV Encoding + XOR*. This is the base FV encoding algorithm enhanced with the decorrelator on sender side and correlator on receiver side.
(3) *FV Encoding + XOR + Equal*. This is the complete encoding algorithm including the base FV encoding algorithm with the two enhancements of XORing values and performing equality test.

The results are shown in Figure 10. On an average, the first configuration that uses only the base FV encoding scheme provides nearly 13% reduction in switching activity. The second configuration that uses the base FV encoding scheme and the XORing of values, on an average, doubles the reduction in switching activity to nearly 26%. This is consistent with our previous observations. Recall that, on an average, half of the frequent value occurrences are preceded by frequent value occurrences while the other half are preceded by nonfrequent values. The base FV encoding scheme reduces the switching for the former category of frequent value occurrences while the XOR reduces switching for the latter category of frequent value occurrences.

The complete encoding algorithm does outperform the above configurations. On an average, it achieves 30% reduction in switching activity. Therefore, overall the equality test reduces switching by a small additional amount. However,

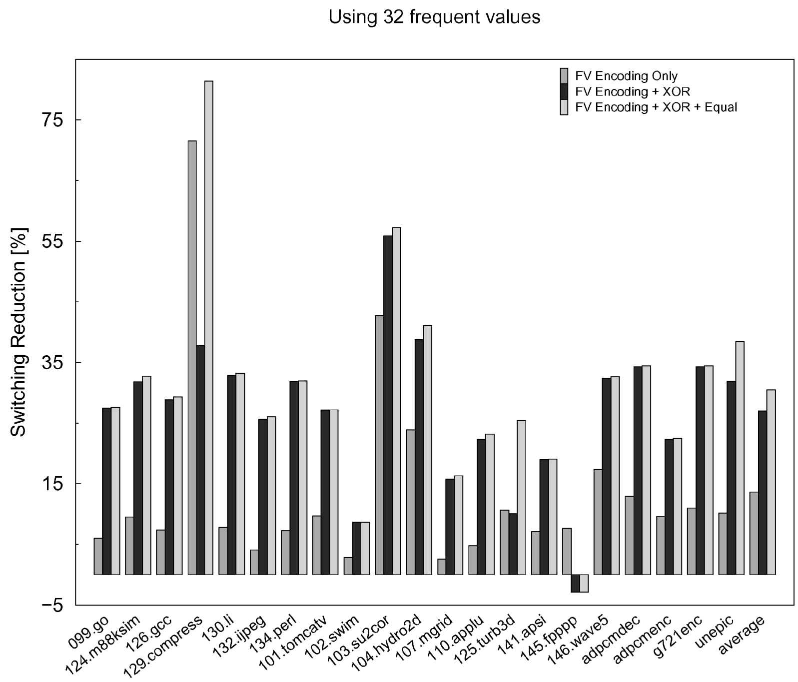


Fig. 10. Effectiveness of FV encoding and its enhancements.

for some benchmarks, the equality test is crucial for obtaining good performance. For the `compress` and `turb3d` benchmarks, the equality test provides a significant increase in performance because sequence of equal values are encountered very frequently. In fact, as we can see, the switching reduction obtained using of the final configuration is more than twice that of the reduction achieved using the second configuration. In fact, in both these cases using the FV encoding scheme alone gives better performance that additional XORing of values.

Next, we considered the impact of excluding frequent values based upon hamming distance between the frequent values and their encoding. The following three pairs of configurations of the encoding algorithm were considered:

(1) *FV Encoding vs. FV Encoding + Exclusion*. This is the comparison of the base FV encoding scheme with and without exclusion of values 0 through 16 as candidates for frequent values (i.e., these values are never added to the frequent value table).
(2) *EV Encoding + XOR vs. FV Encoding + XOR + Exclusion*. This is the base FV encoding algorithm enhanced with XORing of values. The two versions compared are ones with and without exclusion of values 0 through 16 from the frequent value table.
(3) *FV Encoding + XOR + Equal vs. FV Encoding + XOR + Equal + Exclusion*. This is the based FV encoding enhanced with both XORing of values and equal test. The two versions compared are ones with and without exclusion

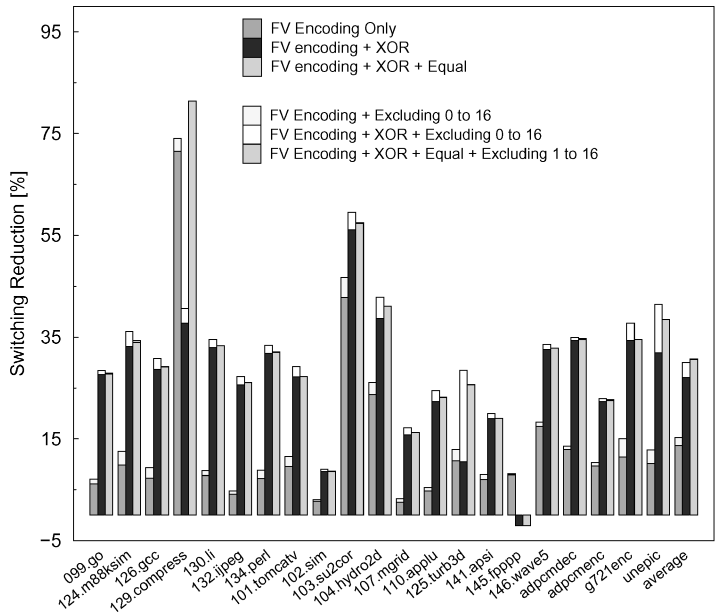


Fig. 11. Impact of excluding values on switching reduction.

of values 1 through 16 from the frequent value table. Recall that the equal test requires hardwiring the value 0 into the frequent value table. It is for this reason only values 1 through 16 are excluded from the frequent value table.

Figure 11 shows that the reduction in switching activity is slightly improved for the first two algorithms. This is because some values that now reside in the frequent value table more often replace small values with very few high bits. However, the performance of the third algorithm is unchanged. That is, the FV encoding algorithm enhanced with XORing and equal test performs equally well with or without exclusion of values. In the remainder of this section, for all experiments involving measurement of reductions in switching activity, we use the FV encoding algorithm with XORing and equal test as the basis for experimentation.

4.2 Encoding and Decoding Activity

The on-chip overhead of performing encoding and decoding is dominated by the accesses to the frequent value table which involves associatively searching for the values and in case a value is not found, the frequent value table is updated using the LRU replacement policy. In this section we compare the performance of various algorithms from the perspective of this on-chip overhead.

4.2.1 *Equal Test and Hamming Distance Based Exclusion of Frequent Values.* Two of the six variations of encoding algorithms that we considered in the preceding section, namely the base FV encoding scheme and FV encoding with XORing, access the frequent value table for each value that is transmitted

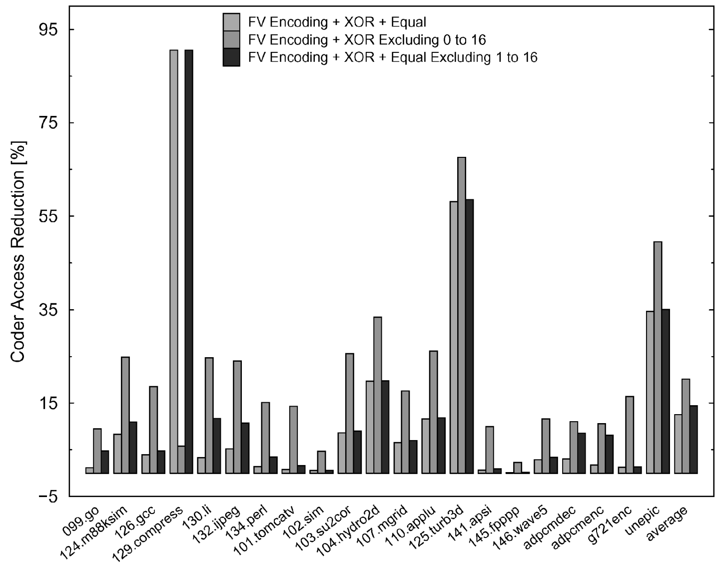


Fig. 12. Impact of excluding values on encoding/decoding operations.

over the bus. However, the remaining four algorithms which use the *equal test* or *exclusion of values* or both avoid accesses to the frequent value table. This is because in both of these cases the frequent value encoding and decoding processes is bypassed. We measured the reduction in accesses to the frequent value table that these four algorithms achieve over the other two algorithms. The results of this study are presented in Figure 12. Two of the FV encoding algorithms, the 4th and the 5th in Figure 11, introduce the same encoder access reduction since they both filter out values 0 through 16. We plot the algorithm with XORing to represent both of them in Figure 12. As we can see, these reductions are substantial. The exclusion of values significantly reduces the accesses performed by base FV encoding and FV encoding with XORing. The reduction in accesses due to the equal test is generally less than that achieved by the exclusion of values 0 through 16 for these algorithms. However, for the `compress` and `turb3d` benchmarks the equal test reduces the accesses to the frequent value table dramatically.

From the perspective of reducing switching activity on the data bus, the two algorithms that perform equally well are one which uses all of our techniques (i.e., FV Encoding, XORing, equal test, and excluding values 1 through 16 from frequent value table) and the one that uses all techniques except that of excluding values. However, as expected, the former algorithm performs, on an average, slightly fewer accesses to the frequent value table than the latter algorithm. Therefore we can conclude that the algorithm which uses all of the techniques, that is, FV encoding, XORing, equal test, and exclusion of values performs the best overall.

4.2.2 *Using Fixed Set of Frequent Values.* As mentioned earlier, an alternative to dynamically identifying frequent values is to identify them first during

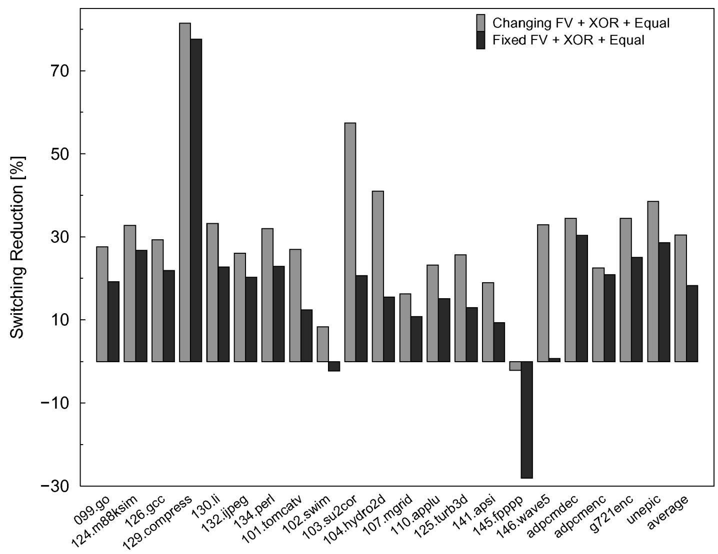


Fig. 13. Changing frequent values versus fixed frequent values.

a profiling run and then use these fixed values during all future program runs. This fixed frequent value set approach avoids spending of energy on updating the frequent value table. We compared the reduction in switching that can be obtained using fixed frequent values with that obtained using the dynamic algorithm described in this paper. The results are presented in Figure 13. As we can see, the reductions using dynamically detected frequent values is significantly greater. On average, using enhanced changing FV encoding scheme we obtain 30% reduction in switching activity while a fixed FV encoding the reduction is only 18%. Moreover, for several of the benchmarks, including `su2cor`, `hydro2d`, `fpppp` and `wave5`, the difference is dramatic. Therefore this approach to reducing on-chip overhead is not very attractive.

4.3 Varying the Number and Width of Frequent Values

4.3.1 *Varying Number of Frequent Values.* We also investigated the effect of the encoder and decoder size on performance by varying the number of frequent values allowed. Our encoding algorithm can be applied to the entire data bus width if the maximum reduction in switching is desired. It can also be applied to a subset of bus wires when minimum hardware expense is demanded. Minor changes are needed when only a subset of bus wires are encoded. For example, assume that only the first 8 bus wires are involved in encoding. Both the encoder and the decoder have only 8 entries. A full value is taken into the encoder and if it is encoded successfully, a code is sent out with respect to those 8 wires. The rest of the wires always carry a zero for encoded values. If the value is not encoded, the original value is sent along the bus and the coders update their content accordingly. The receiver side can resolve both cases in the same way as before without confusion.

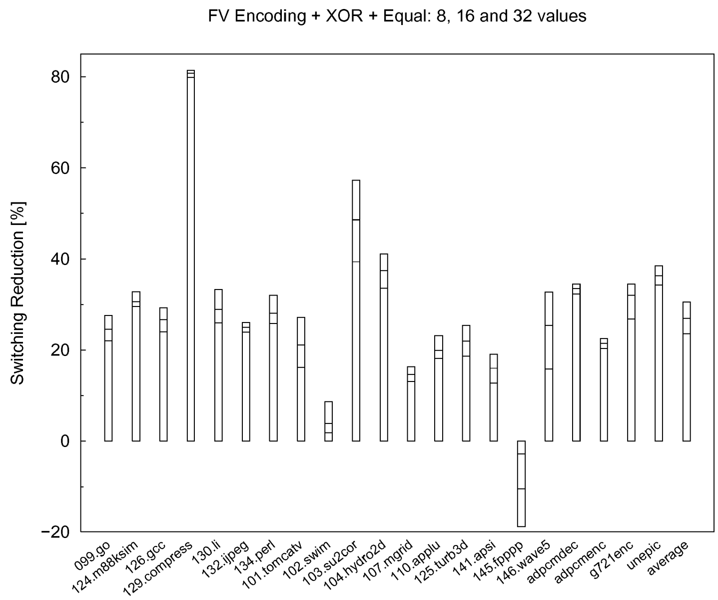


Fig. 14. Varying number of frequent values.

We varied the number of entries in the coders as 8, 16 and 32. The results are given in Figure 14. In some benchmarks, including `tomcatv`, `su2cor` and `wave5`, it can be clearly seen that reductions in switching activity increase significantly with an increase in allowable number of frequent values. In other benchmarks the vast majority of reductions can be achieved simply be using 8 frequent values. The average reduction increases from 23% for 8 values to 30% for 32 values.

4.3.2 *Byte Level Encoding*. So far, we have discussed encoding and decoding schemes based on word level frequent values. In many multimedia benchmarks, where data is operated in unit of bytes, *frequent byte values* may be more abundant than *frequent word values*. We can easily adapt our scheme to handle frequent byte values so that it is friendly to multimedia benchmarks as well. To do this, we simply encode each byte in the 32 bits independently. The 32 entries in the frequent value table can be distributed among the four byte positions, that is, 8 frequent byte values are maintained corresponding to each byte position. In other words, the original word level encoder and decoder are broken into 4-byte level encoder and decoder, each with its own decorrelator and correlator.

The performance of frequent byte encoding as compared to frequent value encoding depends upon program characteristics. For example, if more frequent bytes are found than frequent words, byte level encoding may out perform word level encoding. On the other hand if the benchmark contains mostly frequent words, byte level encoding will hurt performance. This is because now the frequent word would be split into four frequent bytes each of which will require one high bit during its transfer.

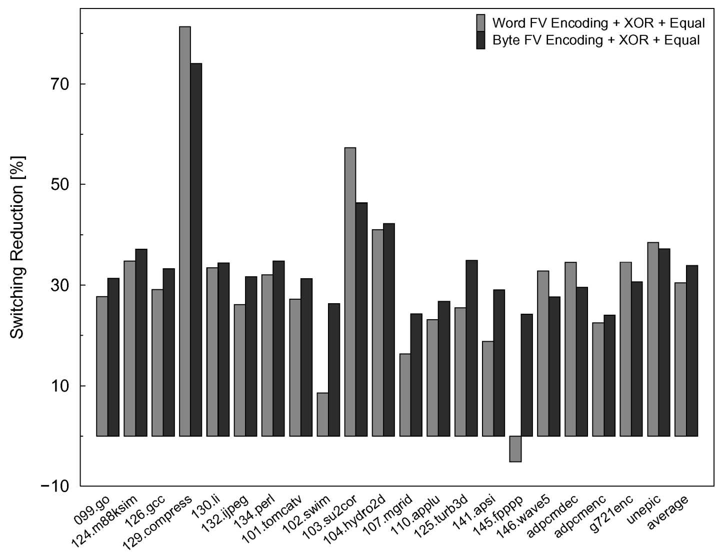


Fig. 15. Byte level frequent value encoding.

The results in Figure 15 shows the gain and the loss of using byte level encoding. For a program like `fpppp`, which has a high level of frequent bytes, the performance is dramatically improved using byte level encoding—instead of 5% increase in switching we now observe a 24% reduction in switching. On the other hand, for a benchmark like `compress` for which the word value encoding performs very well, byte level encoding does not perform as well as word level encoding. On average, a little improvement of 3% was obtained using byte level encoding.

4.4 Accuracy of Frequent Value Identification

4.4.1 *Approximate LRU versus Perfect LRU Replacement.* The identification of frequent values is based upon an approximate LRU policy which uses a timestamp. The size of the timestamp can be varied to achieve different levels of LRU replacement accuracy. Intuitively larger timestamps should provide a better estimation of the least frequently used information and thus perform well during replacement. We performed an experiment in which we compared the encoding rates when using a one bit timestamp (i.e., approximate LRU) with the encoding rates obtained when using an unlimited sized timestamp (i.e., perfect LRU). The results of this experiment shown in Figure 16 disproved our intuition as it shows that a timestamp as small as one bit can perform as well as an unlimited timestamp.

The reason behind the above result is as follows: The data transfered on the data bus between CPU and memory is due to transfer of *cache lines* that contain multiple words. Although these words flow through the encoder and decoder one by one, since they arrive at the bus in close succession, there is little or no difference in their timestamp values. Therefore, even the perfect LRU replacement policy cannot differentiate between these values. Picking any

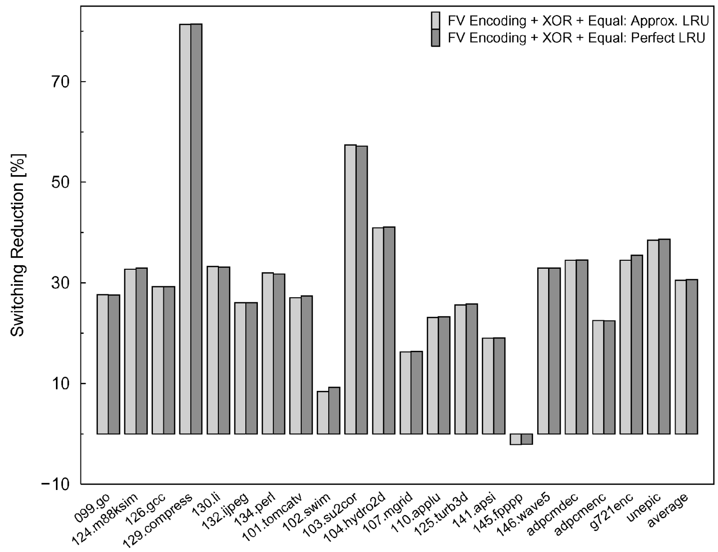


Fig. 16. Comparison with perfect LRU.

one of them may not actually yield a best result. Our result shown in Figure 16 proves that a coarse timestamping method is sufficient in practice.

4.4.2 *Approximate LRU versus Optimal Replacement.* We also conducted another experiment to see how close does approximate LRU come to optimal replacement. We implemented an *optimal replacement policy* in which we *replace the entry that will not be used for the longest period of time in the future*. The optimal policy will yield best switching reduction because it will guarantee the highest hit rate in the encoder. To implement this scheme we ran each program twice. We collected the value trace during the first run and used it in the second run to carry out optimal replacement. Every time we need to perform replacement, we go into the value trace to find the frequent value in the table that appears furthest in the future in the value trace. This is a slow process since the value traces we collected were extremely long. Therefore, we conducted this experiment only for a subset of the benchmarks (11 out of 22). The results presented in Figure 17 show that on an average the switching reduction by our LRU implementation with one bit timestamp is exceeded by the optimal policy by around 11% which is quite reasonable. This is because the optimal policy that we compare with is an offline policy and therefore no online policy will be able to perform nearly as well.

4.5 Performance Without On-Chip Cache

In all of the experiments described so far we assumed that there was an 8K byte on-chip instruction and an 8K byte on-chip data cache. We also repeated our experiments without on-chip caches. The architecture is sketched in Figure 18. This is because in many embedded and DSP processors from AT&T Microelectronics, Motorola, Zilog and Texas Instruments there is no on-chip cache.

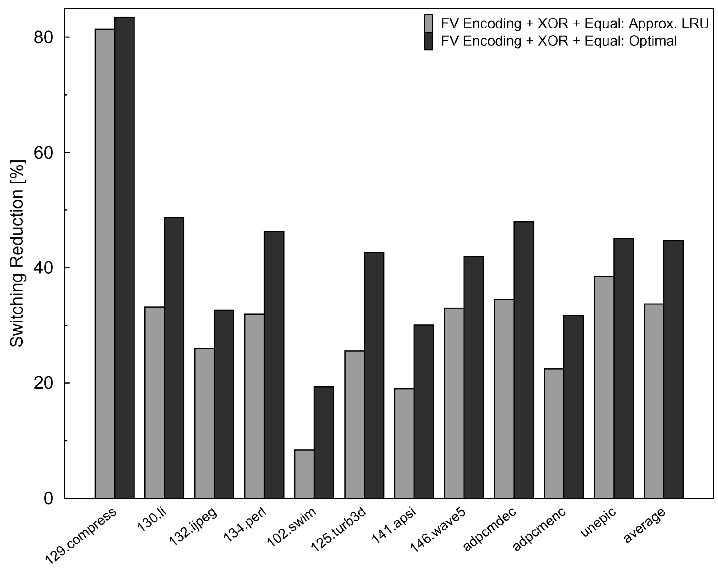


Fig. 17. Comparison with optimal replacement policy.

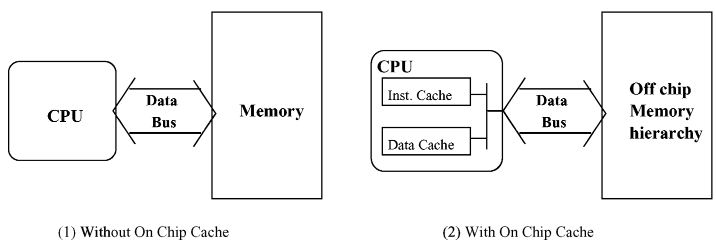


Fig. 18. Architecture models with and without on-chip caches.

The results in Figure 19 show that in the absence of an on-chip cache the reductions in switching activity are even greater. The average reduction for 32 values increases from 30% to 49%. The performance improvement is brought by the data locality within cache lines, which was caught by instruction/data caches and is now being exploited by our encoder and decoder.

5. HARDWARE DESIGN AND ENERGY-DELAY MEASUREMENT

So far, we have measured the effectiveness of FV encoding in terms of switching activity reduction. In this section we develop a detailed hardware design of the encoder to estimate the energy savings and delay introduced due to encoding. We develop a circuit level implementation of an FV encoder with equality test. The decoder has a symmetric structure as the encoder. For simplicity, we will double the energy and delay spent by the encoder to account for total overhead due to FV encoding. Our measurements give an upper bound on the coding energy and delay since the actual decoding process is simpler than the encoding (FV CAM indexing vs. lookup).

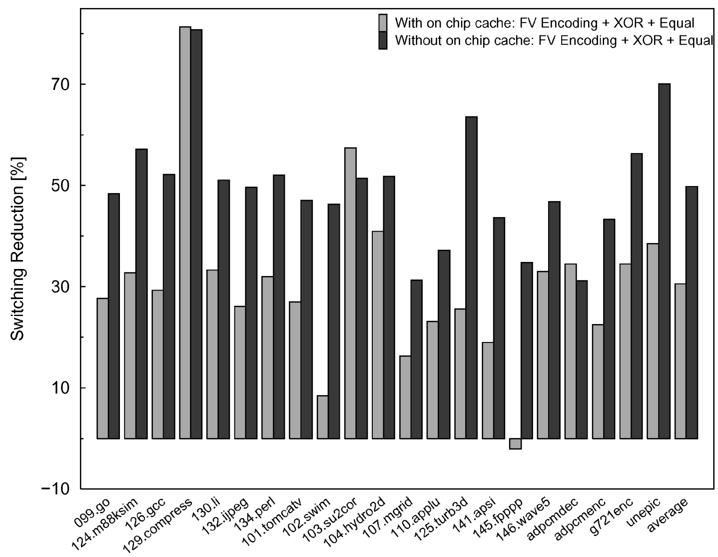


Fig. 19. FV encoding performance with on-chip cache versus without on-chip cache.

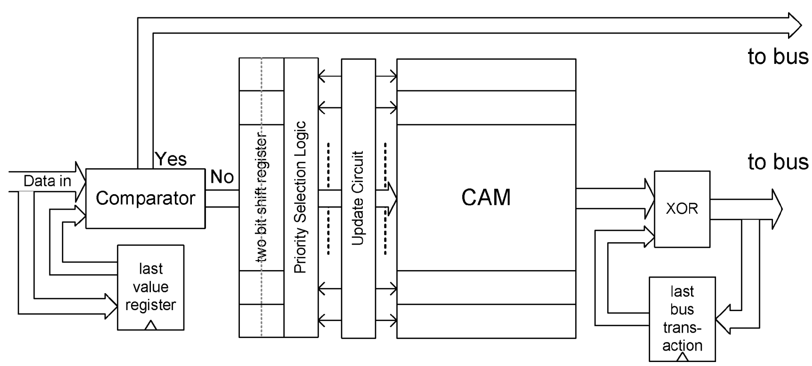


Fig. 20. Encoder components.

5.1 Encoder Design

The overall design of the encoder consists of three main components shown in Figure 20: the *comparator* to implement the equality test (note that value elimination based FV encoding can also be implemented similarly since it also involves a comparator as its first step); *timestamp management unit*; and the *CAM update unit*. Next we present the detailed designs of these units.

5.1.1 *The Comparator.* The comparator is used to compare the current data value and the last data value. Its advantages have been demonstrated in Section 2.2. The comparator is used on every data value. On a success, all the rest of the FV encoder is bypassed and the value is sent directly to the bus. Thus, when total energy is calculated, the comparator energy is charged every time but the rest of the encoder energy is charged only on comparison failures. The circuit for the comparator is shown in Figure 21. Here the D_i stands for the

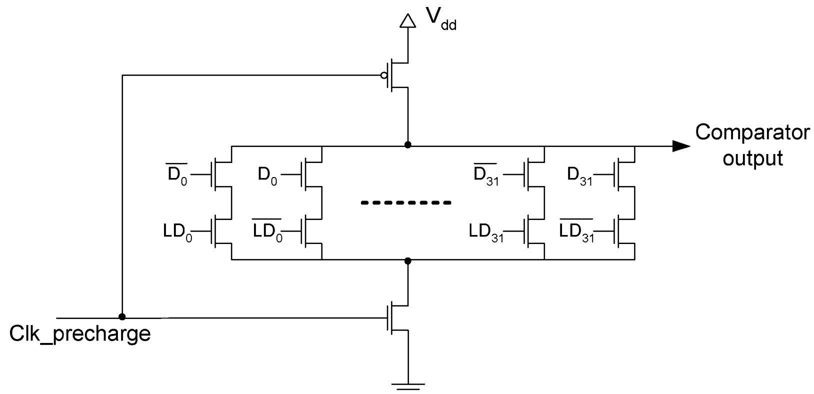


Fig. 21. Circuit for the comparator. D_i stands for the ith bit of the current data value. LD_i stands for the ith bit of the last data value.

ith bit of the current data value and the LD_i stands for the ith bit of the last data value which is supplied from the last value register shown in Figure 20.

5.1.2 *Timestamp Management Unit.* The next component in Figure 20 is the timestamp management unit which includes a two-bit shift register file and a priority selection logic. The first bit in a two-bit register serves as the reference bit and the second serves as the timestamp. At regular intervals, both bits are shifted right as described in Section 3.1. When a miss occurs during a FV CAM lookup, an entry with the smallest two-bit register value is selected. Implementation of this selection turned out to be the most difficult part in the entire FV encoder since finding the smallest value may involve sorting logic. To avoid expensive sorting operations, we observed that for the two-bit registers, there must be an entry containing “00” if the shifting is done once every 16 (or fewer) values that are passed. If there are multiple “00”s, picking out one is enough. Therefore, we define the shift interval as 16, and design the selection logic to find one register that contains “00”. We designed the priority selector such that the first “00” from the bottom is selected from a group of four two-bit registers (see Figure 22). Having a larger group size causes instability in the circuits. Thus, for a 32-entry CAM, eight priority selectors are necessary at the first level. The second level needs two (see Figure 23). The 32 NOR gates attached to each register are to identify value “00”s (true if t_i is 1). The out_i signals determine which t_i's are selected after the first level of priority logic. Thus, there is only one out_i that is 1 for every group of four t_i's. After the second level of selection, the Sel_i determines which group is selected for each group of four out_i's. Finally, the *lo* and *hi* signal indicate if there is an entry selected out of the lower and upper 16 entries respectively. If any entry, say the 29th entry, should be selected, then out_{29} and Sel_7 are both high. If at the same time another entry, say the 1st entry, also contains “00”, priority should be given to the bottom entry. In this case, the *lo* signal is also high, but entry 29 should not be selected. The combination of the outputs are used in the CAM update logic which is explained next.

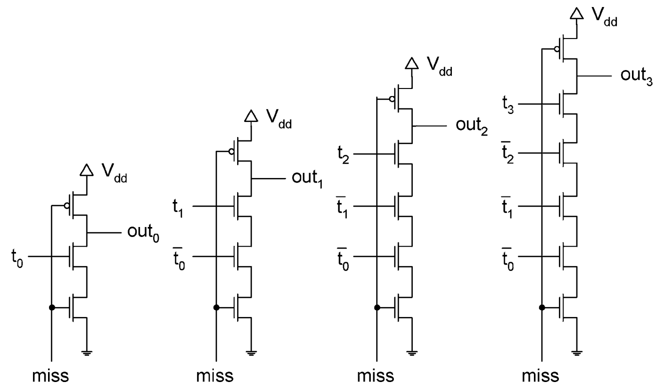


Fig. 22. Single priority selector circuits.

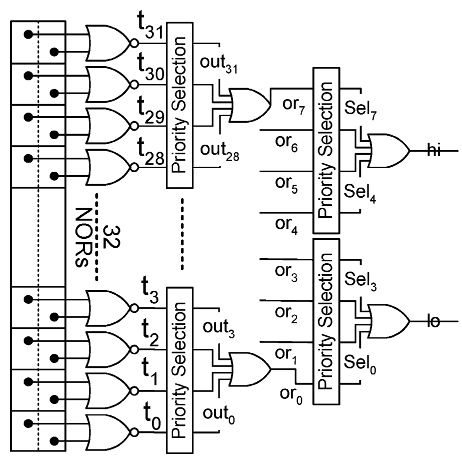


Fig. 23. Timestamp registers and priority selectors circuits.

5.1.3 *CAM Update Unit.* The CAM update logic, shown in Figure 24, is used only on CAM misses. On a miss, a victim entry needs to be selected for replacement. The victim is identified using combinations of the outputs from the priority selection logic. Continuing with the example discussed above, if the 29th and the 1st entry both contain "00" and are selected by both upper and lower group, the input to the 29th CAM entry should be

$$out_{29}\ AND\ Sel_{29}\ AND\ NOT(lo)$$

to give priority to selected lower entry which is the 1st entry in this case. This combination is applied to every entry in the upper half of the CAM with different indices. For the lower half of the CAM entry, the $NOT(hi)$ is not necessary since by default they are given higher priority over the upper half of the CAM.

5.1.4 *CAM Cell.* The most important component of the FV encoder is the FV CAM. In order to generate energy and delay information as accurately as possible, we created an actual layout of a CAM. Figure 25 is the CAM cell circuit

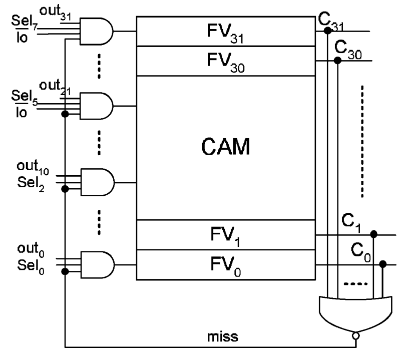


Fig. 24. CAM update circuit.

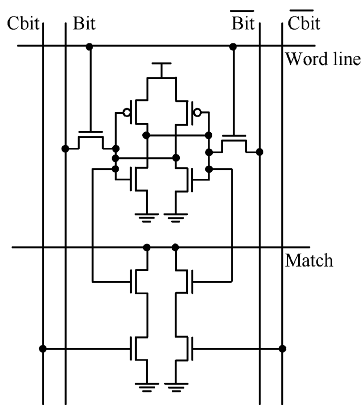


Fig. 25. CAM cell circuit.

that is composed of a conventional six-transistor SRAM cell and dynamic XOR comparators. We used a separate search line *Cbit* from the *Bit* to decrease the search line capacitance and thus the search time. All the *Match* lines are precharged high. On a CAM line miss, the *Match* line is pulled down low generating a logic "0". On a CAM line hit, the *Match* line stays high generating a logic "1". As a result, a CAM miss is determined if all the *Match* lines are low (see Figure 24). A CAM hit results in a single *Match* line that is high with all the rest lines at low, or a natural "one-hot" code.

Figure 26 shows the layout of the CAM cell obtained from the Cadence circuit layout tools [Cadence Corporation]. The technology we used was TSMC 0.18, the most advanced modern CMOS technology available to universities through the MOSIS program [The Mosis Service]. Our ten-transistor CAM cell has dimension of $5.3\ \mu m \times 5.6\ \mu m$. Thus, the entire 32×32 FV CAM occupies $30.39 \times 10^{-9}\ m^2$. The energy and delay information is obtained by using Cadence's Spectra to simulate the net list of the extracted circuits. The results will be presented later.

5.1.5 *XOR Egress.* Lastly, the XOR gates take the inputs from last bus transaction (LD_i) and the encoding result of the current value. The current

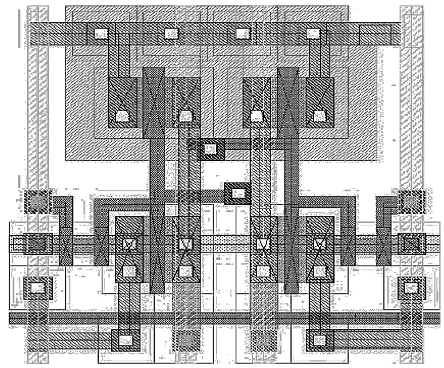


Fig. 26. CAM cell layout.

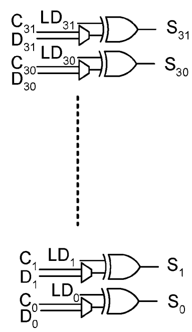


Fig. 27. XOR egress.

result is either the original data value or the output of the FV CAM. For frequent values, the result should be the "one-hot code" generated from the FV CAM (C_i's). Otherwise, it should be the original data value (D_i's). A multiplexer is necessary to choose between the two cases as shown in Figure 27.

5.2 Energy and Delay Results

We used SPICE tool to measure the accurate energy and delay expenditure of the FV encoder logic using 0.18 μ technology (CAM was modeled in Cadence). Since it is infeasible to run benchmarks value traces through SPICE and Cadence tools, we decided to measure energy for each component of the FV encoder and sum up the total energy according to rich statistics collected from benchmark simulations. Table I lists the energy and delay results for each component we discussed earlier.

For components on the main path of encoding, our designs are optimized for speed so that each value can be encoded in a timely fashion. Those components include the comparator, the FV CAM and the xor egress. Other components such as registers are designed using standard implementations. The total delay of the encoding process is less than the sum of the delays listed in the table since many activities can be done in parallel. For example, filling in the last value register can be carried while the value is being compared and encoded.

Table I. Energy and Delay Results for Each Component in FV Encoder

Component	Energy (pJ)	Delay (ns)
comparator	1.277	0.1
registers	0.07 per bit toggle	0.5
priority selection	1.1	0.4
CAM	13.6 per lookup/3.02 per update	0.6/0.8
xor egress	0.24	0.1

Filling the last bus transaction register can be carried with the CAM updating if there is a miss. Thus, there are two major paths in the encoding process. The first one is value encoding path which takes 0.8 ns $(0.1 + 0.6 + 0.1)$. The second one is the CAM update path on a CAM lookup failure. This path takes 1.5 ns calculated from $max(CAM\ lookup\ result\ known,\ priority\ selection\ for\ victim) + CAM\ update\ delay = max(0.1 + 0.6, 0.4) + 0.8 = 1.5\ ns$. Therefore, the critical path is the 1.5 ns updating path. Take a fast system bus such as the 533-MHz bus used in Intel Xeon processor as an example, the FV encoding time for a single value is fast enough to be finished within one bus cycle which is 1.8 ns. The immediate next value can be encoded while the current value is transmitted on the bus, forming a perfect pipelined encoding. Thus, when an entire cache line arrives at the system bus controller the only performance overhead imposed on the transmission is the encoding delay to the *first* value. We experimented with the impact of the encoding delay on overall performance assuming memory latency of 100 cycles and that every cache line sent over the bus incurs four additional *CPU* cycles (2 for encoding and 2 for decoding) assuming two CPU cycles amount to one bus cycle. The average slowdown is 1.0% (maximum is 2.6%) for all the benchmarks tested. This is quite modest considering the amount of bus energy we can save which is shown next.

To obtain the overall energy consumed by FV encoding for each benchmark, we used the following equations:

$$E_{total} = 2 \times E_{FV\_encoder}; \quad (1)$$

$$E_{FV\_encoder} = E_{comparator} \times no.\ of\ values \quad (2)$$

$$+ E_{1-bit\_reg\_toggle} \times (comp.\ input\ toggle\ counts + bus\ toggle\ counts) \quad (3)$$

$$+ (E_{CAM} + E_{priority\_sel} + E_{1-bit\_reg\_toggle} + E_{xor\_egress}) \times no.\ of\ values\ accessed\ CAM \quad (4)$$

$$- E_{CAM\_update} \times no.\ of\ values\ hit\ in\ CAM \quad (5)$$

As mentioned before, the total energy is measured by doubling the FV encoder energy (line 1). Line 2 says that the comparator energy is charged on every value. Line 3 computes the total energy due to storing last value and last bus transaction in registers. Here, energy is charged only when bits are toggled. The fourth line is the energy for values that are not filtered by the comparator and thus flow through the rest of the encoder. The last line says the CAM update energy should not be charged on CAM hits. Using the equations, we calculated the energy spent by the coders.

The results of energy computations are shown in the second column of Table II. The third and fourth column are the total energy spent by the data

Table II. Energy Results for All Benchmarks

Benchmarks	En-/De-coder Energy (mJ)	Bus Energy (mJ) Before	Bus Energy (mJ) After	Energy savings (%)
099.go	0.72	32.27	23.32	25.51
124.m88ksim	0.34	15.42	10.18	31.79
126.gcc	0.02	0.70	0.49	26.89
129.compress	0.02	1.26	0.24	79.69
130.li	0.19	8.08	5.40	30.91
132.ijpeg	0.01	0.71	0.52	24.34
134.perl	0.10	4.90	3.33	29.84
101.tomcatv	0.62	28.32	20.61	25.02
102.swim	0.07	3.81	3.48	6.81
103.su2cor	0.45	22.50	9.60	55.32
104.hydro2d	0.37	17.89	10.55	38.93
107.mgrid	0.40	25.40	21.27	14.71
110.applu	0.31	20.18	15.52	21.55
125.turb3d	0.03	1.19	0.88	22.81
141.apsi	0.11	5.49	4.45	16.91
145.fpppp	0.66	30.93	31.58	−4.24
146.wave5	0.06	3.07	2.06	30.98
adpcmdec	0.00	0.06	0.04	32.91
adpcmenc	0.00	0.04	0.03	20.63
g721enc	0.56	25.61	16.76	32.37
unepic	0.03	1.03	0.63	35.84
Average				28.55

bus before and after FV encoding. We used a typical off-chip bus capacitance of 30 pF, such as the Quad Band Memory technology provided by Kentron Technologies [Kentron Technologies], and assumed the bus voltage is 3.3 V. It should be noted that the proposed FV encoding scheme is suitable for the off-chip bus only since the capacitance of the on-chip buses are significantly smaller so that power savings may not be achieved. The last column of Table II gives the energy savings in percentages. The results clearly show the FV encoder and decoder energy is a negligible factor comparing with the bus energy. The switching activity is the dominant factor in bus energy as the saving percentages are consistent with the switching reductions we showed earlier in the paper. On average, we achieved 28.55% of the energy savings on data buses. We also tested the results for different bus capacitances. If a 10 pF bus was considered, we can still save 24.5% of energy on average. For a 60 pF bus, our saving is 29.6% on average. Therefore, our FV encoding is a very effective way of reducing energy consumption on data buses.

6. RELATED WORK

There has been a significant amount of research done on reducing address bus switching, based on the sequentiality of program counters [Benini et al. 1997; Cheng and Pedram 2000; Su et al. 1994] and regularity of memory accesses [Musoll et al. 1997]. The work that applies to data buses falls in two categories:

(a) general purpose 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.

6.1 Comparison with the General Purpose Bus-Invert Coding Scheme

A well-known general technique for reducing switching is the bus-invert coding scheme [Stan and Burleson 1995b]. 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 enhanced FV encoding for data buses. The results in Figure 28 shows that on an average bus-invert scheme reduces switching by 13.4% and 9.6% in presence and absence of on-chip cache respectively. In contrast the enhanced FV encoding with 32 changing values reduces switching by 30.5% and 49.8% in presence and absence of on-chip cache respectively. Thus, the enhanced FV encoding scheme provides 2 to 4 times greater reduction in switching than bus-invert coding method.

6.2 Comparison with Other Data Bus Encoding Techniques

Some of the work in this category [Ramprasad et al. 1999; Benini et al. 1999] starts from statistical properties of the data streams and compute codes such that value pairs with higher probability of occurrence lead to fewer switching transitions. In Ramprasad et al. [1999], the authors introduce a generic encoder-decoder architecture model and provide a few sample solutions in each module of the generic model. In Benini et al. [1999], the 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 29 compares our enhanced FV encoding with 32 dynamically changing set of values with the adaptive method in Benini et al. [1999]. As we can see, on an average, the reduction in switching achieved using FV encoding is 1.5 to 3 times of that achieved using the adaptive method (9.45% vs. 30.5% with on chip cache and 20.5% vs. 49.8% without on chip cache).

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

7. CONCLUSIONS

In this article, we have demonstrated that by exploiting the characteristic of frequently transmitted values, we can design the FV encoding scheme that reduces

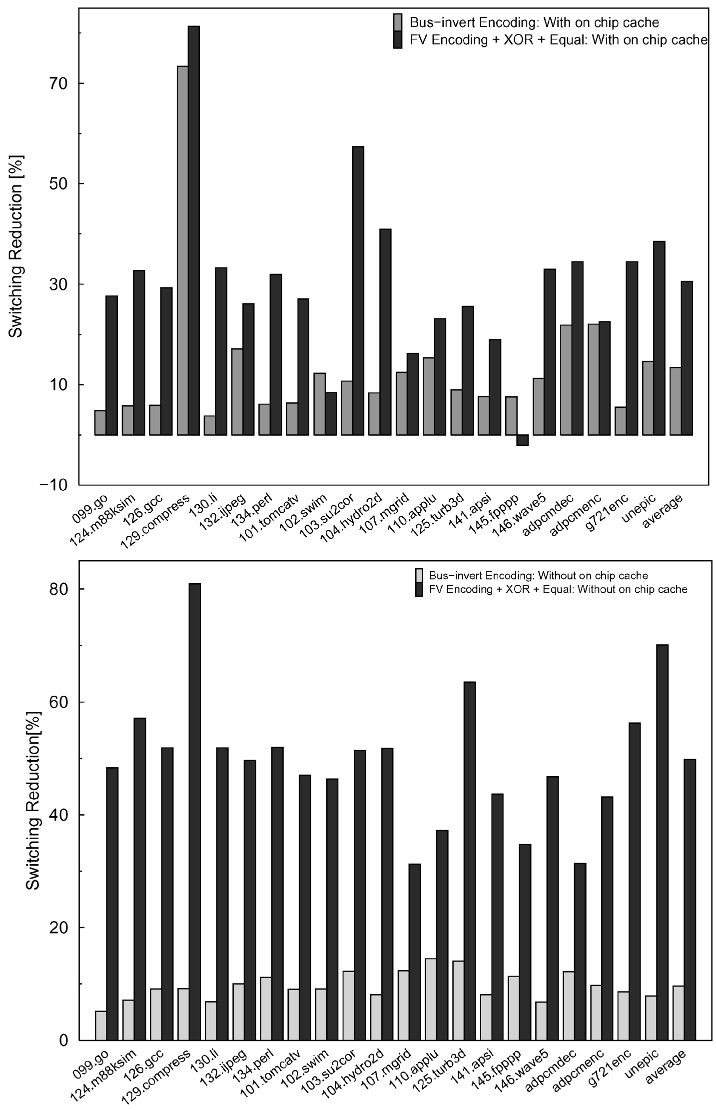


Fig. 28. Bus-invert versus enhanced FV encoding.

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 hardwaremechanism. Our online frequent value identification algorithm compares quite favorably with an offline optimal algorithm. By allowing the set of frequent values to change during execution we obtain reductions in switching and energy that are substantially greater than reductions achieved by a scheme that uses fixed set of frequent values for the entire execution. Finally we have demonstrated that FV encoding outperforms

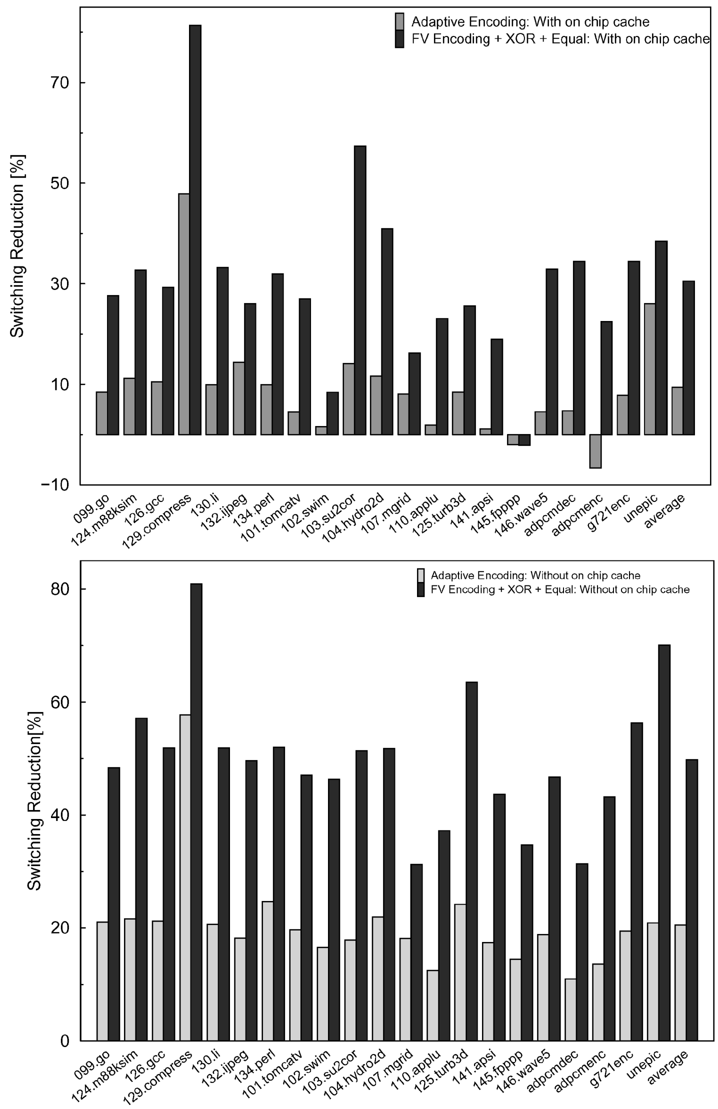


Fig. 29. FV versus adaptive encoding.

both bus-invert coding [Stan and Burleson 1995a] and the adaptive scheme of Benini et al. [1999].

REFERENCES

BENINI, L., DEMICHELI, G., MACII, E., SCIUTO, D., AND SILVANO, C. 1997. Asymptotic zero-transition activity encoding for address busses in low-power microprocessor-based systems. In *Proceedings of the Great Lakes Symposium on VLSI*. 77–82.

BENINI, L., MACII, A., MACII, E., PONCINO, M., AND SCARSI, R. 1999. Synthesis of low-overhead interfaces for power-efficient communication over wide buses. In *Proceedings of the Design Automation Conference*. ACM, New York, 128–133.

CADENCE CORPORATION. http://www.cadence.com.

CHANG, N., KIM, K., AND CHO, J. 2000. Bus encoding for low-power high-performance memory systems. In *Proceedings of the Design Automation Conference*. ACM, New York, 800–805.

CHENG, W.-C. AND PEDRAM, M. 2000. Power-optimal encoding for DRAM address bus. In *Proceedings of the International Symposium on Low Power Electronics Design*. ACM/IEEE, New York, 250–252.

KOMATSU, S., IKEDA, M., AND ASADA, K. 2000. Bus data encoding with adaptive code-book method for low power IP based design. In *Proceedings of the International Workshop on IP based design and Synthesis*.

KRETZSCHMAR, C., SIEGMUND, R., AND MUELLER, D. 2001. Auto-optimizing bus encoding for reduced power dissipation in dynamically reconfigurable hardware. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms*.

KENTRON TECHNOLOGIES. Quad band memory (QBM): matching the fastest systems processors by doubling DDR memory bandwidth. http://www.kentrontech.com/PDF_Files/Presentations/.

RAMPRASAD, S., SHANBHAG, N. R., AND HAJJ, I. N. 1999. A coding framework for low-power address and data busses. *IEEE Transactions on VLSI Systems 7*, 2 (June), 212–221.

THE MOSIS SERVICE. http://www.mosis.com.

MUSOLL, E., LANG, T., AND CORTADELLA, J. 1997. Exploiting locality of memory references to reduce the address bus energy. In *Proceedings of the International Symposium on Low Power Electronics Design*. ACM/IEEE, New York, 202–207.

STAN, M. R. AND BURLESON, W. P. 1995a. Coding a terminated bus for low power. In *Proceedings of the Great Lakes Symposium on VLSI*. 70–73.

STAN, M. R. AND BURLESON, W. P. 1995b. Bus-invert coding for low power I/O. *IEEE Transactions on VLSI Systems 3*, 1 (March), 49–58.

SU, C.-L., TSUI, C.-Y., AND DESPAIN, A. M. 1994. Saving power in the control path of embedded processors. In *Proceedings of the Design and Test of Computers*. IEEE, Computer Society Press, Los Alamitos, Calif., 24–30.

YANG, J. AND GUPTA, R. 2002. Frequent value locality and its applications. *ACM Trans. Embed. Comput. Syst. 1*, 1 (Nov.), 79–105.

ZHANG, Y., YANG, J., AND GUPTA, R. 2000. Frequent value locality and value-centric data cache design. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, New York, 150–159.

Received January 2002; revised May 2003; accepted March 2004