Energy-Efficient Encoding Techniques for Off-Chip Data Buses

DINESH C. SURESH, BANIT AGRAWAL, JUN YANG, and WALID NAJJAR
University of California, Riverside

Reducing the power consumption of computing devices has gained a lot of attention recently. Many research works have focused on reducing power consumption in the off-chip buses as they consume a significant amount of total power. Since the bus power consumption is proportional to the switching activity, reducing the bus switching is an effective way to reduce bus power. While numerous techniques exist for reducing bus power in address buses, only a handful of techniques have been proposed for data-bus power reduction, where frequent value encoding (FVE) is the best existing scheme to reduce the transition activity on the data buses.

In this article, we propose improved frequent value data bus-encoding techniques aimed at reducing more switching activity and, hence, power consumption. We propose three new schemes and five new variations to exploit bit-wise temporal and spatial locality in the data-bus values. Our techniques just use one external control signal and capture bit-wise locality to efficiently encode data values. For all the embedded and SPEC applications we tested, the overall average switching reduction is 53% over unencoded data and 10% more than the conventional FVE scheme.

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: Design, Performance, Measurement.

Additional Key Words and Phrases: Low-power data buses, encoding, bus switching, internal capacitances

ACM Reference Format:

Suresh, D. C., Agrawal, B., Yang, J., and Najjar W. 2009. Energy-efficient encoding techniques for off-chip data buses. ACM Trans. Embedd. Comput. Syst. 8, 2, Article 9 (January 2009), 23 pages. DOI = 10.1145/1457255.1457256 http://doi.acm.org/10.1145/1457255.1457256

1. INTRODUCTION

Power dissipation is a critical design criterion for embedded systems and especially for mobile computing devices [Semiconductor Industry Association 2003]. These devices often draw their current from batteries that place a limited amount of energy at the system's disposal. Consequently, reduced power and

Authors' addresses: D. C. Suresh, B. Agrawal, J. Yang, and W. Najjar, Department of Computer Science and Engineering, University of California, Riverside; email: najjar@cs.ucr.edu.

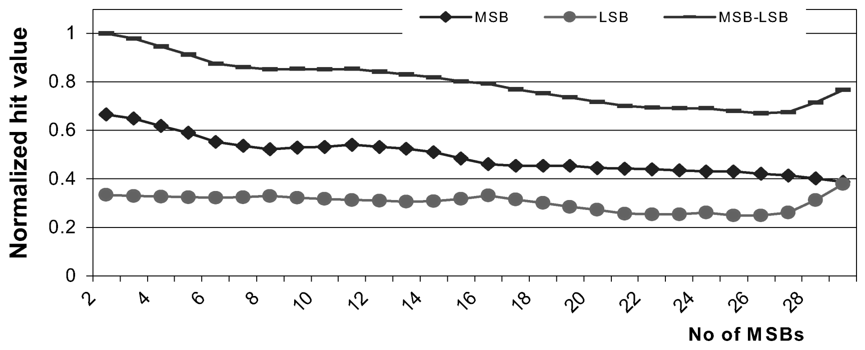


Fig. 1. Average normalized hits for FV, MSB and LSB.

energy consumption of embedded devices translates to longer battery lives and reduced cooling requirements.

Off-chip and On-chip bus lines in VLSI circuits are associated with very large capacitances and they are a major contributor to a system's total power consumption. The power consumption in the bus drivers is in direct proportion to the product of the average number of signal transitions and the line capacitance. Capacitive load of off-chip buses is orders of magnitude larger than that of internal switching nodes [Chern et al. 1992; Givargis and Vahid 1998; Weste and Eshraghian 1998]. At the expense of a small internal energy cost bus-encoding schemes encode bus values to significantly reduce the bus power during off-chip-transmission.

Both address and data streams are amenable to encoding. Efficient schemes to exploit the sequential and stride behavior of instruction address streams have been proposed [Basu et al. 2002; Benini et al. 2000; Givargis and Vahid 1998; Stan and Burleson 1995]. However, the difficulty in encoding off-chip data bus values lies in the fact that off-chip data streams are less regular.

In off-chip data traces, the high order bits (MSBs) and the low order bits (LSBs) of a data value occur a lot more frequently than the entire data value. For example, if the value 80485678 occurs 10,000 times, then one could safely assume that 8048 occurs at least 10,000 times in the high-order bits, and the value 5678 occurs at least 10,000 times in the low-order bits of data values. In other words, partial data value locality is at least as abundant as data value locality. We propose that, besides storing the values encountered in the recent past, the high- and low-order bits of the values should also be stored in separate tables. The intuition behind doing so is that, for every repeating data value, there might be many nonrepeating data values that contain the same high- or low-order bits. Throughout this article, we will refer to repeating values as frequent values (FV) and nonrepeating values as nonfrequent values.

Figure 1 shows the average normalized hits for the MSB, LSB, and the MSB-LSB portions of data values for the benchmarks in NetBench, MediaBench, and the SPEC2000 application suites. A scheme that encodes both the MSB and LSB portions besides encoding the entire data values can efficiently exploit the large hit rate in the MSB and LSB portions. Throughout this article, we will refer to this scheme as the FV-MSB-LSB scheme.

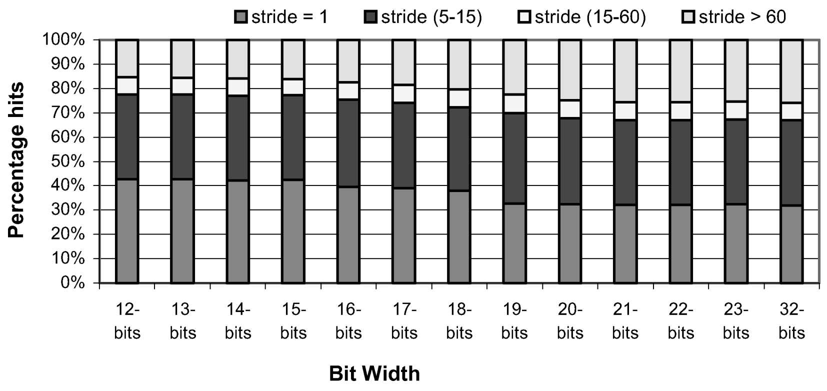


Fig. 2. Common strides of repeating values in Jpegdecode benchmark.

To better understand the need for caching the entire as well as partial data values we evaluated the following: for each repeating data value, we measured the number of cycles it takes for the value or its portion to recur. Figure 2 summarizes our results for the *Jpegdecode* benchmark [Burger and Austin 1997]. The x-axis shows the bit-width of the values under consideration and the y-axis shows the percentage contribution of values occurring within k-cycles (k = 1, 15, 60, and >60). For entire data values (32-bits wide), 30% of the frequent values repeat within the next cycle. When a value repeats in consecutive cycles, it does not result in any off-chip bus transitions even in the absence of an encoding scheme. Hence, encoding such occurrences of frequent values would not yield any significant energy savings. Nearly 33% of the hits shown in Figure 2 repeat within a span of 15 cycles. Encoding such values would yield higher savings. About 20% of the repeating data values recur after 60 cycles. Data values of smaller widths repeat more frequently than data values of larger width. Hence, encoding schemes that can capitalize on temporal locality in both entire and partial data values would be highly effective in reducing the energy consumption.

In this article, we propose three novel data bus-encoding schemes to reduce power consumption in the off-chip data buses. *FV-i* encoding extends the frequent value encoding (FVE) [Yang and Gupta 2001] scheme to maintain a larger window of recently encountered data values and is, hence, capable of encoding more data values than the original FVE scheme. *FV-i-MSB-j* encodes both entire data values and the most significant bits (MSB) portions of repeating data values. *FV-MSB-LSB* encodes entire data values as well as the MSB/LSB portions of data values. The FV-MSB-LSB scheme provides an average energy reduction of 53% over unencoded data and yields an additional 10% improvement in energy on top of FVE.

The remainder of this article is organized as follows. In Section 2, we discuss the related work. In Section 3, we describe our data bus-encoding schemes and its variations. In Section 4, we describe our experimental framework. In Section 5, we evaluate the energy consumption of our schemes and the energy consumed in the off-chip bus. In Section 6, we analyze the impact of our encoding schemes on performance of the system, and in Section 7, we conclude.

2. RELATED WORK

Data bus-encoding schemes, like bus-invert coding [Stan and Burleson 1995] adaptive coding [Komatsu et al. 1999], and FVE [Yang et al. 2004; Zhang et al. 2000], do not assume any prior knowledge of the application. A scheme that operates without prior knowledge of input data is highly desirable because in many application-domains, knowing the data in advance might prove to be a very stringent requirement.

Bus-invert coding [Stan and Burleson 1995] transfers a data value either in its original form or in its complement form depending on whose hamming distance with the previous bus transmission is smaller. An external complement signal is used to let the destination know that the value sent on the bus is in one's complement form, and hence, it should not be interpreted as is. It is a simple method that assumes values are uniformly distributed across the entire value space.

The adaptive encoding scheme [Benini et al. 1997], taking the next step further, is capable of online adaptation to the value streams by learning the statistics on the fly. As collecting the accurate statistics for the value streams can be very expensive, the proposed adaptive encoding operates bit-wise rather than word-wise. Thus, it loses the correlation among the bits within a single value.

Gray code encoding [Su et al. 1994] capitalizes on the observation that consecutive values are often sent during successive bus cycles. If gray code was used for representing addresses, sending consecutive values would result in only one transition on the bus. In T0 encoding 3, an external control signal is used to indicate that the current and previous bus values differ by one, and there is no transition activity in the bus wires while sending the second value. Though these schemes work well with address streams, they do not work well with data streams because sequential data values are rarely sent on successive bus cycles.

Bus expander [Citron and Rudolph 1995] and dynamic base register caching (DBRC) [Ferrens and Park 1991] propose compaction techniques to increase the effective bus-width. DBRC uses dynamically allocated base registers to cache the higher order bits of address values. Ramprasad et al. [1999] applied a general communication model to analyze the bus-encoding schemes. Victor and Keutzer [2001] address the problem of minimizing effect of interwire capacitance by converting a data value sequence into a self-shielding sequence in which no two adjacent bus lines change in opposite directions at the same time. For a 32-bit bus, this scheme needs additional 14 bus lines in order to minimize the cross-talk delay.

Figure 3 shows a symmetric pair of coders that are usually used for bus encoding. An encoder/decoder (codec) is placed at the memory side and the processor side of the off-chip data bus. The codec decides whether the data value should be encoded or not before placing the value on the data bus. When the codec encodes data values, it asserts a control signal to let the destination know that the current value is encoded and hence, it should not be interpreted "as is." The codec design is symmetrical in nature to handle both read and write operations by the CPU. In case of a CPU read, the processor side codec works

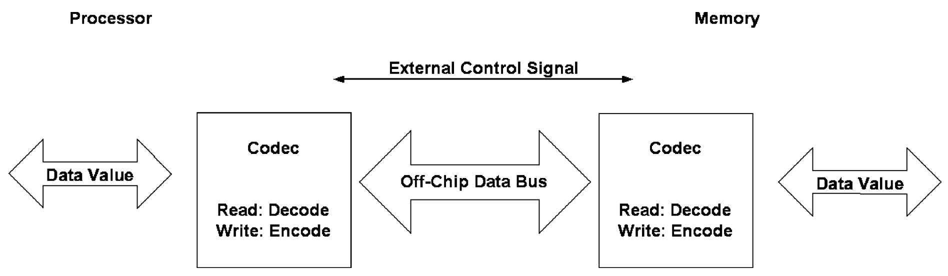


Fig. 3. A symmetric bus codec used in processor to memory communication.

as a decoder; while the memory side codec encodes the value to minimize bus switching. In case of a CPU write, the processor side works as an encoder and the memory side codec works as a decoder.

Table-based encoding schemes exploit temporal locality of data values in order to encode data. In these schemes, tables (or their variants) are used to store recently seen data values or their portions in order to encode/decode data. At the end of every bus cycle, the contents of the encoder and decoder tables are exact replicas of each other. When a new value comes in, the tables are checked to see if the value was encountered in the recent past. If the incoming value was seen before, instead of sending the entire data value, a code corresponding to the value is sent. Most of the table based schemes do a fully associative search on the table entries.

FVE [Yang and Gupta 2001; Yange et al. 2004] is a symmetric table-based scheme that operates in a manner similar to the scheme described above. The FV codec has a k-bit, k-entry table to store previously seen data values. Here, k is the width of the data bus. Before placing a data value on the data bus, the encoder compares the data value with the values stored in the table. A hit in the table implies that the current data value had been encountered in the recent past. In case of a hit, the codec generates a code corresponding to the hit index in the table. The code has the form of "one-hot" code meaning that there is only a single "1," and its position corresponds to the hit index in the table. In the event of a miss in the table, the data value is stored at the encoder and it is then sent over the bus "as is." The decoder checks to see if the data bus value is a one-hot code. If the bus value is a one-hot code, the decoder reads the data value from the table by using the one-hot code as an index to the table. If the data bus value is not encoded, the decoder stores the value in the table and sends the value "as is." Data values are maintained in the table using the LRU replacement policy. This technique has been shown to work very well for off-chip data buses.

Self-organizing list-based encoding [Mamidipaka et al. 2001] minimizes the transition activity between the codes assigned to the most frequent incoming symbols. Their technique efficiently exploits the sequential nature of address streams and the locality of addresses in multiplexed address bus values. Working zone encoding (WZE) [Musoll et al. 1998] keeps track of a few working zones that are favored by the application. Whenever possible, the addresses are expressed as a working zone offset along with an index to the working zone. The encoder and decoder have a few registers to keep track of the working zones and the index selects the current working zone's value from one of the registers. They also extended the WZE scheme for data buses. The working zone offsets are encoded using one-hot codes. However, this technique requires extra bit lines leading to redundancy in space.

LV et al. [2002] proposed a dictionary-based encoding scheme where in the upper few lines of the bus wires are kept in a high-impedance state and the lower bits are encoded. While this scheme works well for large-sized caches, traces with smaller cache sizes often tend to exhibit high entire data value locality. Hence, this scheme fails to exploit the occurrences of entire data values and consequently, the reduction in switching activity is not significantly high. We observed that any scheme that exploits data value locality should be able to exploit entire as well as partial data value locality in order to achieve optimal energy savings.

Our work differs from all of the aforementioned works in the following aspect. Our innovative schemes exploit value locality in full-width data streams as well as partial-width data value streams. The most important difficulty we solved here is not to increase the number of control signals outside the data bus. Our technique uses just one external control signal to indicate the presence of encoded values on the data bus. Our encoding schemes are also capable of maintaining a larger history of data values than the maximum possible history length in the FVE scheme and hence, our schemes have a higher probability of encoding incoming data values in the presence of data locality.

3. PARTIAL- AND FULL-WIDTH DATA BUS ENCODING SCHEMES

In this section, we describe the following three low-power bus-encoding techniques for efficient processor to memory communication.

—FV-i encoding: It is an extension to the FVE scheme [Yang and Gupta 2001]. It encodes entire data values and uses just one external control signal to encode more values than the original FVE.

—FV-i-MSB-j encoding: encodes entire data values or the K-Most significant bits of data values (K is fixed by the designer).

—FV-MSB-LSB: encodes entire data, MSB portion and/or LSB portion of data values.

We would first describe the high-level design methodologies including how to implement a larger value history table size, how to incorporate MSB and LSB values and how the data correlation/decorrelation is done. We then elaborate each technique and its variations in detail.

3.1 Design Methodologies

3.1.1 *Increasing the Table Size.* The FVE scheme sends a one-hot code for a data value, if it is contained in the frequent value table. However, the size of the frequent value table has the following limitation: For a k-bit wide data bus, the number of entries stored in the frequent value table cannot exceed k. Consequently, a value can be encoded only if it is contained in the k-stored entries. By storing more than k entries, one has a higher probability of encoding an incoming data value. However, if we try to encode more values within the framework provided by the FVE scheme, we would require additional external control signals.

Control signals require the availability of a free pin on the chip and are, hence, very expensive to provide. So, we propose a framework that does not increase the number of control signals required by the original FVE (which is one) when we increase the table size. However, increasing the table size does require more number of control signals. The trick is to utilize portion of data bus wires as control signals. For the remainder of this article, we will refer to these data bus wires as *internal control signals*. If the enlarged table size is a multiple of the base table size, the internal control signals can serve as the index to the different portion in the table. For example, a double-sized table needs only one internal control signal to indicate whether the code is generated from the first half or the second half of the table. In this case, the internal control signal reduces the effective base encoding table size by one. The next question is what portion of the data bus should be selected as internal control signals. Through our experiments, we found that the transition activity in the lower-order bits of the data bus is often slightly higher than the activity in the high order bits. Hence, while sending encoded values, we choose to make the least significant bits as control signals. By doing so, they would not contribute much to the total switching. In summary, our proposed method for increasing the table size can be put formally as follows:

Consider a k-bit wide data bus. In order to keep a history of more than k, k-bit values, the number of entries stored in the table is of the form $(k - m) \times 2^m$, where m represents the number of internal control signals. Using the first $k - m$ lines, we send a one hot code corresponding to i mod (k-m) where i is the hit index in the enlarged frequent table. The last m lines, along with the index transmitted on the bus, are used to specify the position of the data within the table. For this scheme, the maximum number of transitions while sending any encoded value is $m + 1$.

3.1.2 *Bit-Width of Stored Values.* As stated earlier, besides storing entire data values, we also store the MSB and LSB portions of the data value in separate tables. In order to determine the optimal width of the MSB and LSB entries, we varied the bit-width of the entries from 2 to 29 bits in steps of 1 and observe the switching reduction for each case. The results are provided in Section 6. Since we encode table hits using one-hot code, the number of table entries should be equal to the bit-width of the stored entries. However, as shown in Section 3.1.1, we can maintain more entries in the FV, MSB, and LSB tables using internal control signals.

3.1.3 *Correlator/Decorrelator.* We use a correlator/decorrelator in all of our encoding schemes. At the encoder's end, the correlator XORs, the current data value with the previous data bus value. The correlator's output is placed on the data bus. At the receiving end, decorrelator XORs, the current and previous data bus value to obtain the current data value. Presence of a

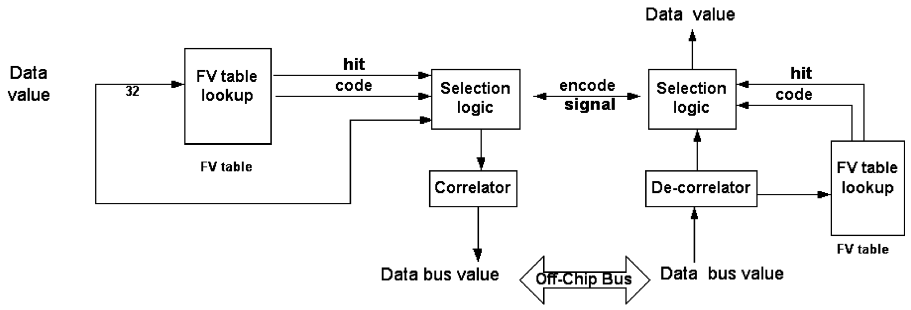


Fig. 4. Codec structure for FV-i encoding scheme.

correlator/decorrelator combination ensures that an off-chip bus wire toggles only for those bit-positions in which the data has a binary value of 1.

In the following subsections, we explain our encoding schemes in detail.

3.2 FV-i Encoding

Figure 4 illustrates the operation of an FV-i codec. FV-i scheme overcomes the limitations of FVE by maintaining larger sized tables and can hence, encode more data values. FV-i maintains larger tables using the method described in Section 3.1.1. When i = 0, FV-i scheme becomes the FVE scheme. We evaluate the performance of FV-i scheme for three values of i: 0,1, and 2. For a 32-bit wide data bus, the number of table entries for FV-1 and FV-2 are 62 ((32-1) $\times$ 2) and 120 ((32-2) $\times$ 4), respectively.

3.2.1 *FV-i Encoder.* The encoder receives the data value from the processor/memory, and it decides whether the data should be encoded before it gets placed on the off-chip data bus. For every incoming data value, the encoder looks up the FV table to check for past occurrences of the data value. The selection logic sees the output of the tables and decides whether the data should be encoded or not. If the selection logic decides to encode the data, it asserts the encode signal and declares the encoded data as the current data bus value, else it lowers the encode signal and sends the data value “as is.” The data bus value passes through the correlator before it finally gets placed on the data bus.

3.2.2 *FV-i Decoder.* The decoder can receive encoded or unencoded data from the data bus. The data bus value passes through the decorrelator and then reaches the selection logic. The selection logic checks encode signal to see if the data is encoded. If the data is unencoded, it is forwarded as is to the processor/memory. Otherwise, using the one-hot code contained in the encoded portion plus the internal control signals, the selection logic picks up the data value from the FV table to construct the decoded value. The decoded value is then forwarded to the processor/memory.

3.3 FV-i-MSB-j Encoding

In this scheme, in order to encode both the entire data and its MSB portion, we have two tables: FV table and an MSB table. The FV table stores the entire data

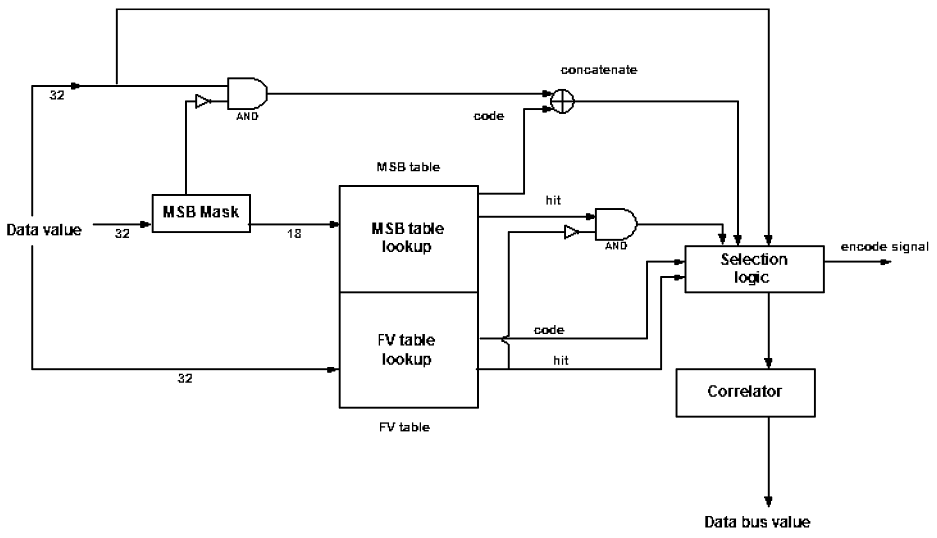
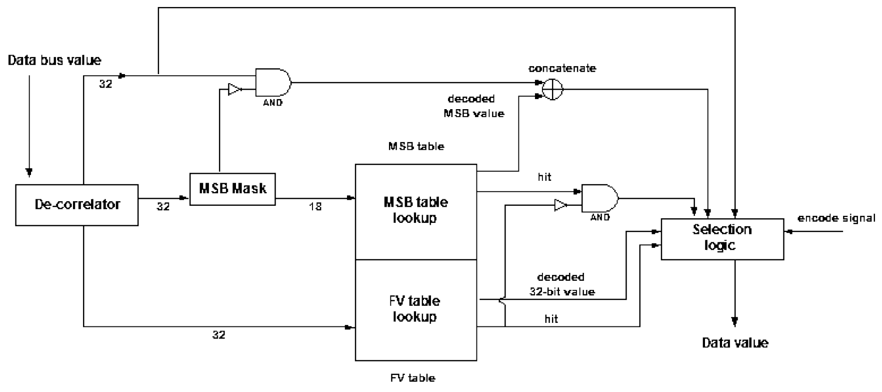


Fig. 5. Encoder structure for the FV-i MSB-j scheme.

value while the MSB table stores the "r most significant bits" of an incoming data value. Here, r is a number that is fixed by the designer and is subject to the constraint that $r < k$, where k is the data-bus width. We evaluate the performance of three instances of this scheme: FV-0-MSB-1 (FVMSB), FV-1-MSB-2, and FV-2-MSB-2. i and j refer to the factors by which the appropriate tables are increased. For example, If we store k-bit wide and r-bit wide entries in the FV and MSB tables, respectively, then FV-1-MSB-2 scheme would have k-entries in the FV table and 2r-2 entries in the MSB table while FV-2-MSB-2 scheme would have 2k-2 entries in the FV table and 2r-2 entries in the MSB table. Here, FV-1-MSB-2 uses the r^{th} bit position as internal controls signal for the MSB table. Likewise, FV-2-MSB-2 uses the k^{th} bit line and the r^{th} bit line as internal control signals for the FV and the MSB tables, respectively. The following paragraphs illustrate the codec's operation as an encoder and a decoder.

3.3.1 *FV-i-MSB-j Encoder.* Figure 5 shows the FV-i-MSB-j encoder. For every incoming data value, the encoder looks up the FV and MSB tables to check for past occurrences of the entire data value and the MSB portion, respectively. In the event of a hit in both tables, the FV hit takes precedence. The selection logic sees the output of the tables and decides whether the data should be encoded or not. If the selection logic decides to encode the data, it asserts the encode signal and declares the encoded data as the current data bus value, else it lowers the encode signal and sends the data value "as is." The data bus value passes through the correlator before it finally gets placed on the data bus.

3.3.2 *FV-i-MSB-j Decoder.* Figure 6 illustrates the operation of an FV-i-MSB-j decoder. The decoder can receive encoded or unencoded data from the data bus. On an incoming data bus value, the selection logic checks the encode signal to see if the data are encoded. If the data are unencoded, it is forwarded



(⊕ = concatenate operator)

Fig. 6. FV-i-MSB-j decoder.

as is to the processor/memory. Otherwise, using the one-hot code contained in the encoded portion, the selection logic picks up the data value from one or more of the stored tables to accurately reconstruct the decoded value. For example, if the selection logic finds that there is a hit in the MSB table only, and then it picks up the MSB portion from the table and uses the LSB portion of the data "as-is" to get back the data value. The decoded value is then forwarded to the processor/memory.

Figure 7 illustrates the encoder algorithm for the FV-2-MSB-2 scheme. This scheme uses upper and lower FV tables to store entire data values. The least significant bit (0^{th} bit) is used as an internal control signal to select one of these two tables. Likewise the r^{th} bit position is used to choose between the upper and the lower MSB tables. Figure 8 illustrates the operation of FV-i-MSB-j encoding with an example. In the third transaction, even though there is a hit in the MSB table, the data value is sent unencoded. This is because the lower portion of the data value is zero in all of its bit positions. Consequently, the encoder would generate a 32-bit one-hot code for the MSB hit. When the decoder receives the 32-bit one-hot code, it would decode the bus value as a 32-bit frequent value. By choosing not to encode such MSB table hits, FV-i-MSB-j scheme can effectively encode the data using just one external control signal.

3.4 FV-MSB-LSB Encoding

The FV-MSB-LSB scheme aggressively encodes incoming data values by sending one-hot codes for the entire data value, the MSB portion, and the LSB portion whenever possible. To accomplish this, FV-MSB-LSB uses three tables: a FV table, MSB table, and an LSB table. The following paragraphs illustrate the codec's functionality as an encoder and a decoder.

3.4.1 *FV-MSB-LSB Encoder.* Figure 9 illustrates the operation of a FV-MSB-LSB encoder. For every incoming data value, the encoder looks up the FV, MSB, and LSB tables to check for past occurrences of the entire data value, MSB portion, and the LSB portion, respectively. In the event of a hit in multiple tables, the FV hit takes precedence. If the selection logic is informed to encode

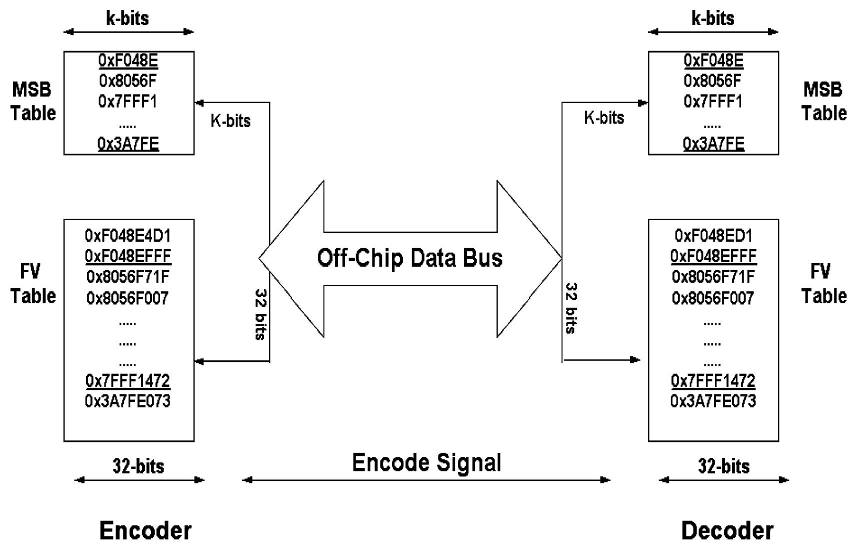
```
for each data value do
    if data value not in U-FV TABLE nor L-FV TABLE nor U-MSB TABLE
            nor L-MSB TABLE then
        encode signal = 0  /* value not in any of the tables */
        send data unencoded
    else
        encode signal = 1
        if hit in U-FV TABLE then                /* FV-hit is in the first k-1 entries */
            send one-hot code
            $0_{th}$ bit of data bus = 0
        else
            if hit in L-FV TABLE then            /* FV-hit is in the next k-1 entries */
                send one-hot code
                $0_{th}$ bit of data bus = 1
            else
                if U-MSB TABLE hit then     /* MSB hit is in first r-1 entries */
                    if  LSB != 0 then
                        encode high-ordered bits (one-hot code)
                        $R_{th}$ bit of data bus = 0
                        send low-ordered bits unencoded
                    else
                        encode_signal = 0
                        send data unencoded.
                    end if
                end if
                if L-MSB TABLE hit then /* MSB hit is in the next r-1 entries */
                    if LSB  != 0 then
                        encode high-ordered bits (one-hot code)
                        $R_{th}$ bit of data bus = 1
                        send low-ordered bits unencoded
                    else
                        encode_signal = 0
                        send data unencoded.
                    end if
                end if
            end if
        end if
    end if
end for
```

Fig. 7. Encoding algorithm for FV-2-MSB-2 scheme.

the data, it asserts the encode signal and declares the encoded data as the current data bus value, else it lowers the encode signal and sends data value "as is." The data bus value passes through the correlator before it finally gets placed on the data bus.

Figure 12 shows the FV-MSB-LSB encoder algorithm. An FV hit is always encoded. When there is a miss in the FV table, we have the following possibilities:

—hit in both tables (encode both portions)
—hit in one of the tables
—miss in both (send data unencoded)



Encoder						Off-chip bus		Decoder				
#	Data Value	FV CAM	MSB CAM	i-FV	i-MSB	Enc. signal	Bus Value	FV CAM	MSB CAM	i-FV	i-MSB	Decoded data
1	0XF048EFFF	**hit**	x	0	0	1	0x40000000	**hit**	x	0	0	0XF048EFFF
2	0XFO48E4CE	miss	**hit**	x	0	1	0x800004CE	miss	**hit**	x	0	0XFO48E4CE
3	0XF048E000	miss	**hit**	x	x	0	unenc.data	-	-	x	x	0XF048E000
4	0X7FFF1472	**hit**	x	1	0	1	0x00000005	**hit**	x	1	0	0X7FFF1472
5	0X3A7FE789	miss	**hit**	0	1	1	0x00003789	miss	**hit**	0	1	0X3A7FE789

Legend
# => Transaction id   x => dont' care   - => No Search operation performed
i-FV => internal control signal for FV table   i-MSB => internal control signal for MSB table

Tables storing m-bit wide values can accomodate upto 2 *(2 ^{m} - i) entries while using 'i' internal control signals. In this example, the FV and MSB CAMs use one internal control signal each and can hence store upt o 62 {2*(32-1)} and 2*(k-1) entries respectively.

1. A hit in the FV CAM is always encoded. Hence the search results in other tables do not matter.
2. There is a miss in FV CAM and a hit in the upper portion of the MSB table. The MSB portion of the data is encoded while the lower order bits are sent "as-is".
3. There is a miss in the FV CAM. The upper portion of the data is found in the MSB CAM. However, the lower potion of data is zero. The encoded data will have one-hot code in the upper bits and the value zero in its lower bits, thereby producing a 32-bit one-hot code. The decoder will decode this encoded value as a 32-bit frequent value. So the data is not encoded and no search operation is performed at the destination.
4. There is a hit in the lower half of the FV CAM. The internal control signal for the FV table is set to 1.The one hot code (0x00000004) is logically OR ed with the internal control signal (0x1) to obtain the current encoded data bus value.
5. There is a hit in the lower half of the MSB CAM and a miss in the FV CAM. The code for the MSB portion (0x00002) is logically OR ed with the MSB's internal control signal (0x1) to obtain the high ordered bits of the encoded value (0x00003). The low ordered bits are sent "as-is".

Fig. 8. FV-i-MSB-j. An example. FV and MSB tables are implemented as content-addressable memories (CAMs).

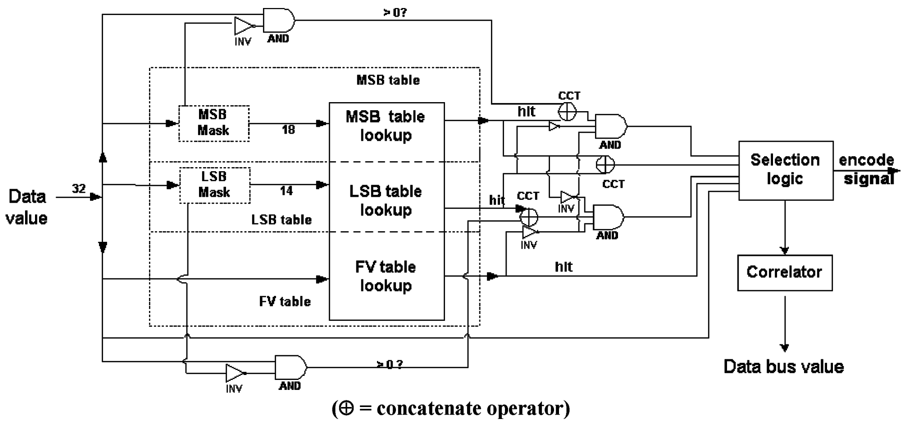


Fig. 9. Encoder for the FV-MSB-LSB scheme.

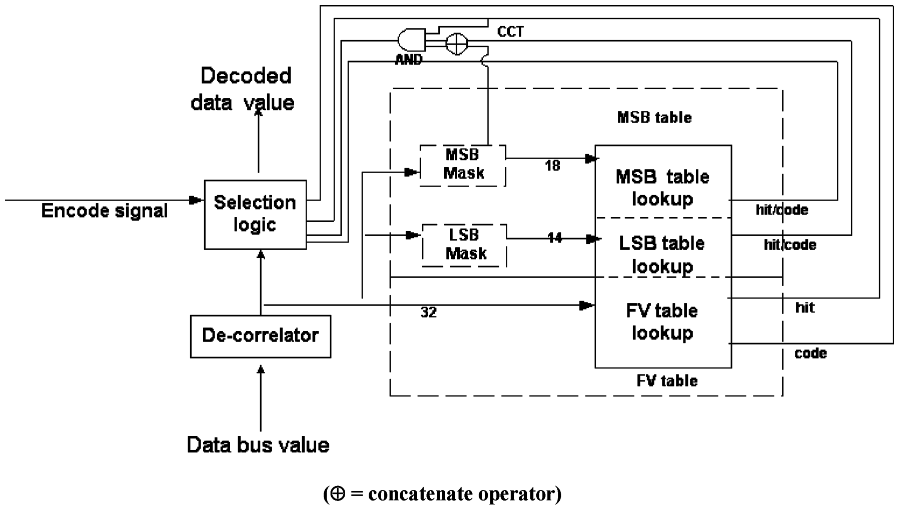
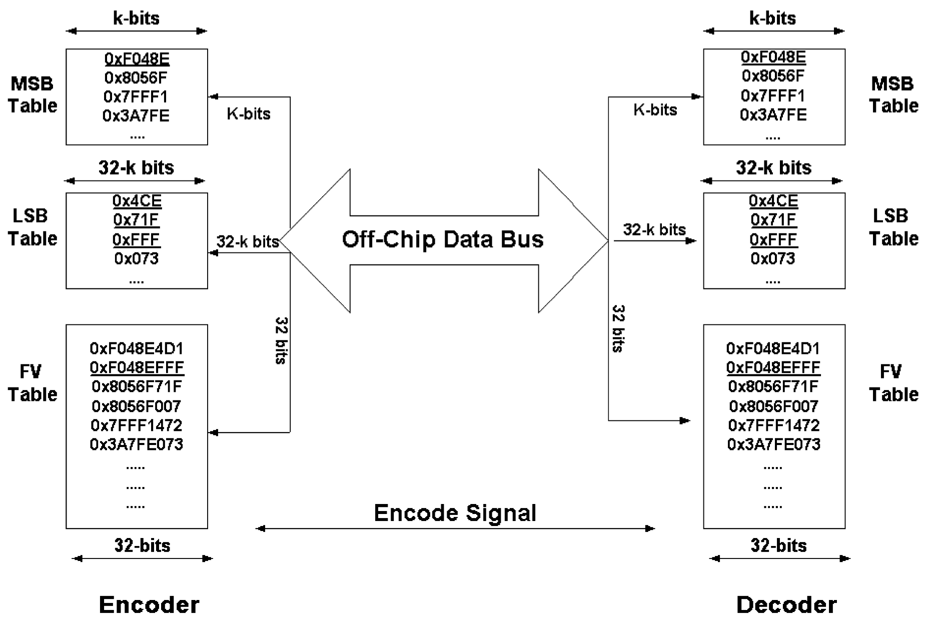


Fig. 10. Decoder for FV-MSB-LSB scheme.

If there is a hit in the MSB table and there is a miss in the LSB table, we check to see if the LSB portion is in the form of a one-hot code. If the LSB portion is nonzero and is not in the form of a one-hot code, we encode the data. In all other cases, the data are sent unencoded. This way, the destination can decode the data value without any ambiguity. We adopt a similar approach during a hit in the LSB table and a miss in the MSB table.

3.4.2 *FV-MSB-LSB Decoder.* Figure 10 shows the operation of an FV-MSB-LSB decoder. The decoder can receive encoded or unencoded data from the data bus. On an incoming data bus value, the selection logic checks the encode-signal to see if the data are encoded. If the data are unencoded, they are forwarded as is to the processor/memory. Otherwise, using the one-hot code contained in the encoded portion, the selection logic picks up the data value from one or more



Encoder					Off-chip bus		Decoder			
#	Data Value	FV CAM	MSB CAM	LSB CAM	Enc. signal	Bus Value	FV CAM	MSB CAM	LSB CAM	Decoded data
1	0XF048EFFF	**hit**	x	x	1	0x40000000	**hit**	x	x	0XF048EFFF
2	0XFO48E4CE	miss	**hit**	**hit**	1	0x80000800	miss	**hit**	**hit**	0XFO48E4CE
3	0XF048E000	miss	**hit**	miss	0	unenc.data	-	-	-	0XF048E000
4	0XF048E777	miss	**hit**	miss	1	0x80000777	miss	**hit**	miss	0XF048E777
5	0X8542E71F	miss	miss	**hit**	1	0x8542E002	miss	miss	**hit**	0X8542E71F
6	0x100004CE	miss	miss	**hit**	0	unenc. data	-	-	-	0x100004CE

Legend
# => Transaction id **x => dont' care** **- => No Search operation performed**

1. A hit in the FV CAM is always encoded. Hence the search results in other tables do not matter.
2. In the event of a miss in FV CAM, a hit in both tables is always encoded.
3. There is a miss in the FV and LSB CAMs. The upper portion of the data is found in the MSB CAM. However, the lower potion of data is zero. The encoded data will have one-hot code in the upper bits and the value zero in its lower bits, thereby producing a 32-bit one-hot code. So the data is not encoded and no search operation is performed at the destination.
4 and 5. hit on only one of the MSB or LSB CAMs. The data is encoded/decoded without ambiguity.
6.The current data value encounters a miss in FV and MSB CAMs. Lower portion of the data is found in LSB CAM. However, the MSB portion of the data is in the form of a one-hot code. If encoded, the destination might interpret the encoded value as a hit in both MSB and LSB CAMs. Hence, such instances are sent unencoded. The destination checks the encode signal and does not perform a search in any of the tables.

Fig. 11. FV-MSB-LSB example. The FV, MSB, and LSB tables are implemented as content addressable memories (CAMs).

of the stored tables to accurately reconstruct the decoded value. For example, if the selection logic finds that there is a hit in the MSB table only, and then it picks up the MSB portion from the table and uses the LSB portion of the data as is to get back the data value. The decoded value is then forwarded to the processor/memory.

Figure 11 demonstrates the operation of FV-MSB-LSB encoder with an example.

```
for each data value do
    if data value not in FV TABLE nor MSB TABLE nor LSB TABLE then
        encode signal = 0    /* value not in any of the tables. */
        send data unencoded
    else
        encode signal = 1
        if hit in FV TABLE then    /* it is a frequent value */
            send one-hot code
        else
            if both MSB and LSB TABLE hit then
                encode high-ordered bits          (one-hot code)
                encode low-ordered bits           (one-hot code)
            else          /* Check for an MSB hit and encode if necessary */
                if only MSB TABLE hit
                    if no of ones in LSB >= 2
                        encode high-ordered bits    (one-hot code)
                        send low-ordered bits unencoded
                    else
                        encode_signal = 0
                        send data unencoded.
                    end if
                end if                                /*Check for LSB hit */
                if only LSB TABLE hit
                    if no of ones in MSB >= 2
                        encode low-ordered bits    (one-hot code)
                        send high-ordered bits unencoded
                    else
                        encode_signal = 0
                        send data unencoded.
                    end if
                end if
            end if
        end if
    end if
end for
```

Fig. 12. Algorithm for FV-MSB-LSB scheme.

4. EXPERIMENTAL SETUP

We modified the sim-outorder simulator in the SimpleScalar toolset [Berger and Austin 1997] for our experiments. For MSB/LSB based schemes, we varied the number of bits captured from 2 to 29 bits in steps of 1. Based on the average reduction in switching activity for different benchmarks, finally we fixed the number of bits to be captured for each scheme.

In order to evaluate the effectiveness of our encoding schemes, we used a wide range of benchmarks that are representative of both embedded and desktop application. Our test programs consisted of benchmarks from the MediaBench [Lee et al. 1997], MiBench [Buthaus et al. 2001], NetBench [Memik et al. 2001], and the SPECINT2000 [Standard Performance Evaluation Corporation 2000] benchmark suites.

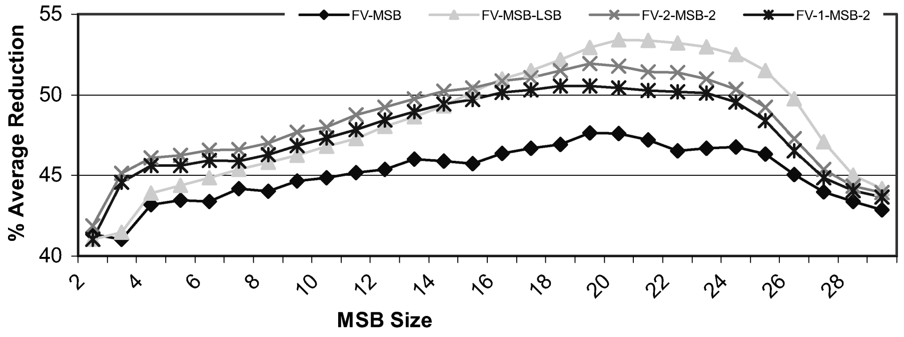


Fig. 13. Average reduction in switching activity for varying MSB lengths.

SPECINT applications are normally run on desktoplike architectures with multiple levels of cache memory. Hence, we opted for a second-level cache while running SPECINT applications. While running embedded system applications, we opted for architecture without L2 cache to closely mirror the commercially deployed embedded systems. We fixed the L1 cache size at 4KB, and we did not include a second-level cache in our simulated architecture while executing embedded applications. We included a 64KB L2 cache while executing SPECINT applications with *ref* dataset. We fixed the block size of the instruction and data caches at 32 bytes. For a 32-bit wide off-chip bus, we assumed the on chip and off-chip latencies of instruction and data caches to be one cycle and 100 cycles, respectively. Figure 13 shows the percentages of average reduction in switching activity for varying bit lengths. Based on the figure shown, we fixed the number of MSB bits for FV-MSB-LSB, FV-2-MSB-2, and FV-1-MSB-2 to be 20, 19, and 20 bits, respectively.

The main objective of our encoding schemes is to reduce the energy consumption in the off-chip bus. Switching activity on the bus indicates the number of times during which the bus lines are charged and discharged and is proportional to the dynamic power dissipated in the off-chip bus. We measure the reduction in off-chip bus-switching activity in order to compute the energy consumption of the bus.

5. ENERGY

5.1 Bus Power Model

We use a bus power model similar to the one discussed by [Catthoor et al. 1998]. In general estimating, the energy used in the off-chip interconnects is difficult. We can approximate the capacitance for the bus using the formula:

$$C_{\text{bus}} = C_{\text{metal}} \times \text{No. of Bus lines.}$$

In this expression, C_{metal} is the capacitance of the metal interconnect for each bus line. Using the numbers given in [Catthoor et al. 1998], it is estimated to be 20 pF. C_{bus} gives the effective capacitive load to be driven during a bus transaction.

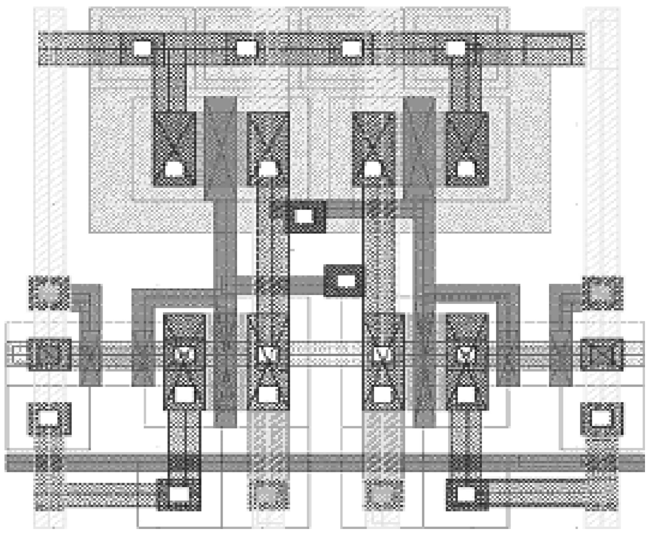


Fig. 14. Layout of the CAM cell used in our design.

5.2 Coder's Energy

In order to determine the energy consumed in the codec itself, we created an actual layout of the CAM cell and other components used in our encoding scheme. In the following paragraphs, we describe each of the codec components in greater detail. Our codec design has four major components: tables, correlator/decorrelator, selection logic, and timestamps.

Tables: These are used to hold the recently encountered data values or their portion and are implemented as CAM cells. Figure 14 shows a CAM cell used in our design. Since the CAM cell circuit is critical in our design, we used two separate search lines: a Cbit line and a Bit line in order to decrease the capacitance associated with the Cbit search line. We used the Cadence layout tools and extracted the circuit from the layout. We used TSMC 0.18μ technology, the most modern CMOS technology available to the universities through the MOSIS program. We simulated the extracted netlists using Cadence's Spectra in order obtain the energy and delay information.

Correlators are implemented as XOR gates. They take the selection logic's output and the previous data-bus value as inputs. At the destination, the current bus value is XORed with the last transaction value to get back the original data. A correlator/decorrelator combination ensures that there is an off-chip bus transition only in those bit-positions that contain a binary value of 1. Figure 15 shows the correlator circuit. P_i denotes the previous bus transaction, C_i denotes the current output of the selection logic, and B_i denotes the current data bus value.

Selection logic: chooses between a table hit and the unencoded data. Figure 16 describes the selection logic for the FV-i-MSB-j scheme. Based on the hits in the FV and MSB tables, the selection logic picks the one-hot code, MSB code or the unencoded data. The energy consumed in the selection logic is the sum of

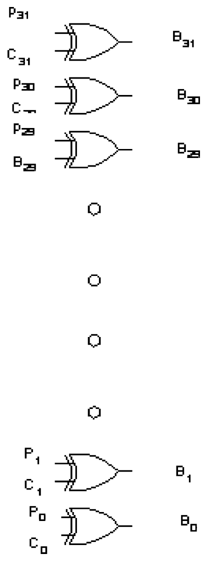


Fig. 15. Correlator.

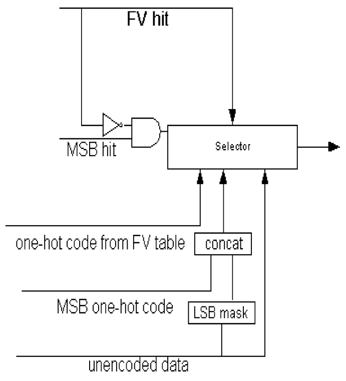
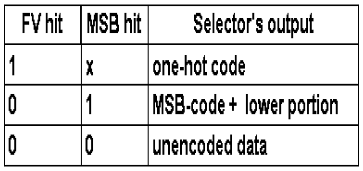


Fig. 16. Selection logic for FV-i-MSB–j.

the energy consumed in the logic gates (LSB mask and MSB hit signal) and the energy consumed by the selector (18×0.095pJ + 1.33pJ = 3.04pJ). Figure 17 shows the selection logic's output for different inputs.

Timestamps: In order to evict stale table entries and to facilitate the accommodation of new table entries, we use a 2-bit timestamp with 1 reference bit. The reference bit is akin to a most significant bit, and it ensures that the most recently accessed table values are not evicted from the table. The timestamps are shifted right every 16 cycles, and the priority selection logic of the timestamp logic picks the value with the least timestamp. We found that the energy consumption in the timestamp entries to be 0.07pJ and the delay associated with the update operation was 0.5ns [Suresh et al. 2003]. The energy consumed by the priority selection logic of the timestamp circuit, as reported in Yang et al. [2004], is 1.1pJ.



FV hit	MSB hit	Selector's output
1	x	one-hot code
0	1	MSB-code + lower portion
0	0	unencoded data

Fig. 17. Input and output values for FV-i -MSB-j selection logic.

Table I. Energy consumed by codec components

Component	Energy	Delay
Selection logic	3.04pJ	0.2ns
XOR gates	0.095pJ/Transition pair	0.1ns
Timestamps	0.07pJ	0.5ns
32-bit, 32-entry table	13.6pJ	0.2ns

For simplicity, we will double the energy and delay spent by the encoder to account for the decoder's energy. Table I shows the energy and delay information for different codec components. The following equation gives the total energy consumption for our encoder:

$$\begin{aligned}\text{Energy}_{\text{encoder}} &= \text{Energy}_{\text{tables}} + \text{Energy}_{\text{timestamps}} + \text{Energy}_{\text{Correlator}} \\ &+ \text{Energy}_{\text{selection\_logic}}.\end{aligned}$$

For our encoding schemes, the total energy consumption in the encoder is the sum of the energy consumed in the FV tables, MSB/LSB tables, timestamps, correlator/decorrelator (XOR gates), and the selection logic. Using the above formula, we calculated the value of Energy$_{\text{encoder}}$ for FV-0, FV-1, FV-2, FV-1-MSB-2, FV-2-MSB-2, and FV-MSB-LSB to be 17.38pJ, 34.65pJ, 52.42pJ, 26.65pJ, 42.65pJ, and 37.88pJ, respectively. We calculated the total bus energy per cycle using the following formula [Weste and Eshraghian 1998]:

$$\text{E}_{\text{total}} = \text{E}_{\text{encoder}} + \frac{\{\text{T}_{\text{r}} \times \ \text{CL} \ \times \text{V}^{\,2}\}}{\text{\# of cycles}} + \text{E}_{\text{decoder}},$$

$$\begin{aligned}\text{where,}\quad T_r &= \text{total number of transitions in the off-chip bus}\\ C_L &= \text{Load capacitance of the off-chip bus line.}\\ V &= \text{Supply voltage.}\end{aligned}$$

Parameters used for the calculation are: $\text{C}_{\text{L}} = 20$ pF and V = 3.3 Volts [Catthoor et al. 1998].

From the equation above, we can infer that even while assuming a modest switching activity of 10 transitions per each bus cycle, the bus energy consumption is 2,000pJ while sending unencoded data. Since our encoding schemes achieve nearly 50% reduction in switching activity, the energy saved during each bus cycle is an order of magnitude more than the codec's energy consumption. Hence, the reduction in energy corresponds to the reduction in switching activity for each bus-encoding scheme. Our codec's energy calculation is highly pessimistic. We assume that the encoder and decoder energy is spent on every

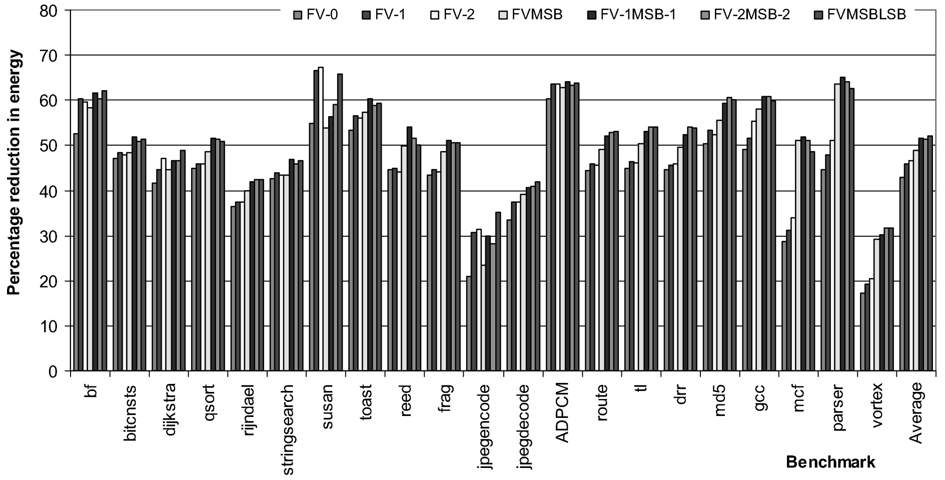


Fig. 18. Average percentage reduction in energy for MediaBench, NetBench, MiBench, and SPECINT applications. *Ref* data set was used while running SPEC int applications.

cycle by calculating worst-case energy consumption in each component of codec. However, while sending unencoded values the selection logic at the decoder end inspects the encode signal and no search operation is performed. Hence, the energy consumption at the decoder end is lesser than the reported energy for encoded values. We use the worst-case power consumption at encoder and decoder ends because we find that worst-case power consumption (activity = 1) for codecs is significantly lower than the power consumption in off-chip bus wires.

We estimate the area of a CAM cell from the layout and it is found to be $11\mu m^2$ in $0.18\mu m$ technology. We estimate the area overhead of CAMs by adding the results of all the CAM segments in $0.18\mu m$ technology and it is found to be $0.021mm^2$, $0.0226mm^2$,and $0.0314mm^2$ for FV-1-MSB-2, FV-MSB-LSB, and FV-2-MSB-2, respectively.

Figure 18 shows the percentage reduction in energy for desktop and embedded applications. For *parser* application, FV-MSB-LSB gives 21% improvement over FVE scheme. For *mcf* benchmark, we get nearly an 18% improvement over FVE scheme. For *Route* and *Jpegencode*, FV-MSB-LSB provides an additional 15% switching reduction on top of FVE. Applications like *parser* and *mcf* are very pointer intensive and are hence highly conducive to MSB-based encoding schemes. For such applications, the MSB-based schemes yield a switching reduction of nearly 20% on top of FVE. On an average, FV-MSB-LSB yields a 10% improvement over FVE scheme. In the following paragraph, we illustrate the area overhead of our schemes.

6. IMPACT ON PERFORMANCE

We achieve significant power savings using the codec, but it comes at the expense of a little performance penalty. The encoding and decoding operations add extra latency in the processor-memory transaction and, hence, there is a slight

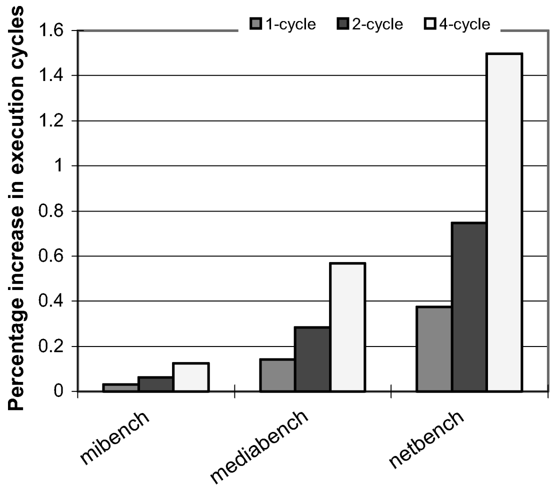


Fig. 19. Average percentage increase in execution cycles for different encoder delays.

decrease in the overall performance. Using the contemporary VLSI technology and the pipelined architecture, the codec can be easily implemented with a delay of two clock cycles, which amounts to a single cycle delay at both the encoder and decoder ends. We take the codec delay to be one cycle, two cycles, and four cycles to evaluate the performance penalty. We instrumented the SimpleScalar simulator to measure the performance penalty for a set of benchmarks and we assumed an off-chip memory latency of 100 cycles. The *average* performance penalty results for embedded system applications for different codec delays are shown in Figure 15. From the figure, we can see that there is almost a double penalty when the codec delay is doubled. On an average, we are incurring 0.06%, 0.29%, and 0.76% performance penalty with a codec delay of two cycles for MiBench, MediaBench, and NetBench, respectively. However, we are achieving 53% energy savings on an average with little performance overhead (i.e., less than 1% among a set of benchmarks).

7. CONCLUSION

Entire and partial data values in off-chip data streams exhibit abundant value locality. We proposed and evaluated three table-based data bus encoding schemes: FV-i, FV-i-MSB-j, and FV-MSB-LSB. Through these table-based bus-encoding schemes, we demonstrated that encoding both entire and partial data values yields significant energy benefits. All of our encoding schemes require just one external control signal. Our schemes make no prior assumptions regarding the input data and are truly dynamic in nature. We tested our scheme on a subset of applications from the MediaBench, MiBench, NetBench, and SPECINT2000 benchmark suites.

For each of our data bus-encoding schemes, we evaluated the codec's energy consumption based on an accurate layout-level description of the codec circuits. FV-MSB-LSB provides nearly 53% energy reduction over unencoded data and nearly 10% improvement over the FVE scheme.

REFERENCES

Basu, K., Choudhary, A., Pisharath, J., and M. Kandemir, M. 2002. Power protocol: Reducing power dissipation on off-chip data buses. In *Proceedings of the 35th Annual IEEE/ACM Symposium on Micro Architecture (MICRO-35)*. IEEE, Los Alamitos, CA.

Benini, L., Macci, A., Macci, E., Poncino, M., and Scarsi, R. 2000. Architectures and synthesis algorithms for power efficient bus interfaces. *IEEE Trans. Comput. Aid. Des. Circ. Syst. 19*, 9.

Benini, L, De Micheli, G., Macci, E., Scuito, D., and Silvano, C. 1997. Asymptotic zero-transition activity encoding for address buses in low-power microprocessor-bases systems. In *Proceedings of the 7th Great Lakes VLSI Symposium,* IEEE, Los Alamitos, CA, 77–82.

Burger D. and Austin T. 1997. The SimpleScalar tool set, version 2.0. Tech. rep. University of Wisconsin-Madison.

Catthoor, E., Wuytack, S., De Gref, E., Balasa, F., Nachtergaele, L., and Vandecappelle, A. 1998. *Exploration of Memory Organization for Embedded Multimedia System Design*. Springer, Berlin, Germany.

Chern, J. H., Jurang, J., Arledge, L., Li, P., and Yang, P. 1992. Multi-level metal capacitance models for CAD design. *IEEE Electron Device Lett. 13*, 32–34.

Citron, D. and Rudolph, L. 1995. Creating a wider bus using caching techniques. In *Proceedings of the 1st International Symposium on High Performance Computer Architecture*. IEEE, Los Alamitos, CA, 90–99.

Farrens, M. and Park, A. 1991. Dynamic base register caching: a technique for reducing address bus width. In *Proceedings of 18th International Symposium on Computer Architecture (ISCA)*. IEEE, Los Alamitos, CA, 128–137.

Givargis, T. and Vahid, F. 1998. Interface exploration for reduced power in core-based systems. In *Proceedings of the 11th International Symposium on System Synthesis*. IEEE, Los Alamitos, CA.

Givargis, T. and Eppstein, D. 2002. Reference-caching using unit distance redundant codes for activity reduction on address buses. In *Proceedings of the 8th International Workshop on Embedded Hardware/Software Codesign (ESCODES'02).* IEEE, Los Alamitos, CA.

Guthaus, M. R, Ringenberg, J. S., Ernst, D., Austin, T. M., Mudge, T., and Brown, R. 2001. MiBench: a free, commercially representative embedded benchmark suite. In *Proceedings of the 4th Annual Workshop on Workload Characterization*. IEEE, Los Alamitos, CA.

Hsiao, I. Y. L., Wang, D. H., and Jen, C. W. 2001. Power modeling and low-power design of content addressable memories. *In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'01)*. IEEE, Los Alamitos, CA, 926–929.

Komatsu, S., Ikeda, M., and Asada, K. 1999. Low power chip interface based on bus data encoding with adaptive code-book method. In *Proceedings of the 9th Great Lakes Symposium on VLSI*. IEEE, Los Alamitos, CA, 368.

Lee, C., Potkonjak, M., and Mangione Smith, W. 1997. MediaBench: a tool for evaluating and synthesizing multimedia and communications systems", In *Proceedings of the 30th Annual International Symposium on Microarchitecture (MICRO-30)*, IEEE, Los Alamitos, CA, 330–335.

Lv, T., Henkel, J., Lekatsas, H., and Wolf, W. 2002. An adaptive dictionary encoding scheme for SOC data buses. In *Proceedings of the Design Automation and Test in Europe.* IEEE, Los Alamitos, CA.

Mamidipaka, M., Hirschberg, D., and Dutt, N. 2001. Low power address bus encoding using self-organizing lists. In *Proceedings of the International Symposium on Low Power Design*. IEEE, Los Alamitos, CA, 188–193.

Memik, G., Mangione Smith, W. H., and Hu, W. 2001. NetBench: a benchmarking suite for network processors. In *Proceedings of the International Conference on Computer Aided Design (ICCAD2001)*. IEEE, Los Alamitos, CA, 39–42.

Musoll, E., Lang, T., and Cortadella, J. 1998. Working zone encoding for reducing the energy in microprocessor address buses. *IEEE Trans. VLSI Syst. 6*, 568–572.

Semiconductor Industry Association (SIA). 2003. National technology roadmap for semiconductors (NTRS). SIA, San Jose, CA.

Raghunathan, A., Jha, N. K., and Dey, S. 1998. *High-Level Power Analysis and Optimization*. Kluwer Academic Publishers, Norwell, MA.

RAMPRASAD, S., SHAMBAG, N. R., AND HAJJ, I. N. 1999. A coding framework for low power address and data buses. *IEEE Trans. VLSI Syst. 7*, 212–221.

STANDARD PERFORMANCE EVALUATION CORPORATION. 2000. SPEC CPU2000 V1.3. http://www.specbench.org/cpu2000.

SU, C. L., TSUI, C. Y., AND DESPAIN, A. M. 1994. Saving power in the control path of embedded processors. *IEEE Des. Test Comput. 11*, 24–30, Volume 11, 1994.

STAN M. R. AND BURLESON, W. P. 1995. Bus-invert coding for low power I/O. *IEEE Trans. Large VLSI Syst., 3*, 49–58.

SURESH, D. C., YANG, J., ZHANG, C., AGRAWAL, B., AND NAJJAR, W. 2003. FV-MSB: A scheme to reduce transition activity on data buses. In *Proceedings of the 10th Annual International Conference on High Performance Computing*. Springer, Berlin, Germany.

VICTOR, B. M. AND KEUTZER, K. 2001. Bus encoding to prevent crosstalk delay. In *Proceeding of the International Conference on Computer-Aided Design (ICCAD)*.

WESTE, N. H. E. AND ESHRAGHIAN, K. 1998. *Principles of CMOS VSLI Design*. Addison Wesley.

YANG, J. AND GUPTA, R. 2001. FV encoding for low power data I/O. In *Proceedings of the International Symposium on Low Power Electronic Design*. IEEE, Los Alamitos, CA, 84–87.

YANG, J., GUPTA, R., AND ZHANG, C. H. 2004. FV-encoding for low power data buses. *ACM Trans. Des. Autom. Embed. Syst. 9*, 3, 354–384.

ZHANG, Y., YANG, J., AND GUPTA, R. 2000. Frequent value locality and value-centric data cache design. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS–IX)*. ACM, New York, 150–159.

Received September 2004; revised February 2006, July 2006, May 2008; accepted August 2008