A Tunable Bus Encoder for Off-Chip Data Buses

Dinesh C Suresh
University of California, Riverside
dinesh@cs.ucr.edu

Banit Agrawal
University of California, Santa Barbara
banit@cs.ucsb.edu

Walid Najjar
University of California, Riverside
najjar@cs.ucr.edu

Jun Yang
University of California, Riverside
junyang@cs.ucr.edu

ABSTRACT

Off-Chip buses constitute a significant portion of the total system power in embedded systems. Past research has focused on encoding contiguous bit positions in data values to reduce the transition activity in the off-chip data buses. In this paper, we propose **TU**nable **B**us **E**ncoding (**TUBE**) scheme to reduce the power consumption in the data buses, which exploits repetition in contiguous as well as non-contiguous bit positions in order to encode data values. We also solve the problem of keeping just one control signal for our codec design.

We compare our results with some of the already existing best schemes such as Frequent Value encoding (FVE) and FV-MSB-LSB encoding schemes. We find that our scheme achieves an improvement of 21% on average and up to 28% on some benchmarks over the FVE scheme and up to 84% over unencoded data. In comparison to FV-MSB-LSB encoding scheme, our scheme improves the energy savings by 10% on average and up to 21% for some media applications at the expense of minimal 0.45% performance overhead. We present a hardware design of our codec and provide a detailed analysis of the hardware overhead in terms of area, delay and energy consumption. We find that our codec can be easily implemented in an on-chip memory controller with small area requirement of 0.0521 mm^2.

Categories and Subject Descriptors
C.4 [**Performance of systems**]: Design studies – *data bus encoding, low power design.*
General Terms: Algorithms, Measurement, Performance, Design, Experimentation,
Keywords: Data bus encoding, Tunable bus encoder, data bus, TUBE

1. INTRODUCTION

Off-chip buses are associated with high capacitance values and hence, while transmitting bus values, they consume a significant portion of the total embedded system power. The energy consumed by the off-chip bus is in direct proportion to the total number of transitions that take place in the bus. Bus encoding schemes are techniques that reduce the bus power consumption by encoding and decoding data prior and subsequent to transmission.

For most of the applications, the transition activity in the off-chip bus wires is non-uniform: some of the bus wires tend to toggle more than the rest. We refer to the set of bits that toggle the least as *silent* bits. Since silent bits remain steady for longer periods of time, the probability of silent bits having the same values in different data values is high. The complementary bit positions of the silent bits incur a lot of transitions and we refer to these bit positions as *hot* bits. Figure 1 shows the silent and hot bit positions in a portion of the data stream. The silent bit positions in the figure are shown in bold. The silent bits incur fewer transitions than hot bits. Hot bits and silent bits are complementary to each other. By encoding both silent and hot bits, the codec can efficiently encode data.

We explore the design space and evaluate the performance of different TUBE codec configurations. We run a set of experiments by executing MediaBench[6], NetBench[9] and SPECINT2000 [12] benchmarks and we find that our scheme achieves an average energy reduction of 66% over unencoded data, which is a 21% improvement over FVE [15]. For some media applications, we achieve as much as 84% reduction in energy over unencoded data. We analyze the impact of the codec on the overall performance of the system and we find that TUBE provides significant energy savings with an overhead of as little as 0.45% of the total execution time.

2. TUBE DESIGN

In this section, we present TUBE design and explain various design parameters, codec algorithm, and hardware design. TUBE uses tables at the encoder and decoder ends. The table consists of various *segments,* where each *segment* extracts a predefined set of bit positions from the data values. Each *segment* consists of a finite set of segment entries and each segment entry, in turn, comprises of a code field and a data field. The code field in the segment entry contains an *M-hot code*. M-hot code is defined as a value whose binary representation has a high value (logic '1') only in M different bit positions, where M is a small number (usually one or two). The data field of the segment entry stores selected bit positions of the incoming data value. For a segment storing codes of width k-bits, containing up to M-hot codes, the maximum number of allowable segment entries is given by

$$\sum_{i=1}^{M} C_i^K = \sum_{i=1}^{M} \frac{K!}{(K-i)!*i!}$$

During the first occurrence of a full or partial bit-pattern, the encoder stores the bit-pattern in its segments and transmits the data

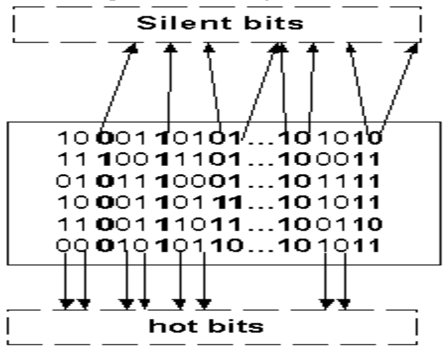

Figure 1 shows silent and hot bits in a data stream. Silent bits are shown in bold.

TUBE Encoder Algorithm

```
For each incoming value
Do
   encode_signal = 1
   lookup value in segments
   if hit in full data segment then
         send code for the hit location
   else
         if hit in other upper segments then
               if hit in lower segments then
                     current_bus_value = upper_code OR lower_code
                     internal_control_signal = 1
                     /* MSBLSB or Silent-hot hit */
               else
                     current_bus_value = upper_code OR lower_data
                     internal_control_signal = 0
               end if
         else
               send data unencoded
               encode_signal = 0
         end if
   end if
Done
```

Figure 2 shows the TUBE encoder algorithm.

value over the data bus without encoding. Upon receiving the unencoded value, the decoder stores the bit-pattern from the data value in its segments. Thus at the end of every bus cycle, the encoder and decoder's table contents are exact replicas of each other For subsequent occurrences of the repeating bit pattern, the codec sends an *M-hot code* for the repeating portion. As used in other techniques [15], a pair of correlator and decorrelator is added to the two ends of the buses. They are inverse functions of each other and their purpose is to reduce the correlation between successive values. The codec associates a three-bit timestamp with the table entries and evicts stale entries using Least Recently Used (LRU) replacement policy. We picked LRU as it can be easily implemented with minimal hardware.

We find the number of simultaneous MSB and least significant bits LSB hits to be significantly higher than the hits in LSB segment only (miss in MSB). Hence, we choose to encode an LSB hit only if it happens to be an MSBLSB hit. Likewise, a hot segment hit is encoded only if there is a hit in the silent segment also. TUBE uses codes of two different widths and these codes are sent in the upper and lower bus wires of the off-chip data bus. The code for the MSB, entire data portion and silent bit segment hits are sent on the upper portion of the data bus while the code for the LSB and hot bits are sent on the lower portion of the data bus. Hence, during an MSBLSB hit or a silent/hot hit, a code is sent in both the upper and lower bus wires. For the remainder of this paper, we will refer to the segments that send code in the upper bus wires as upper segments. Likewise, lower segments are segments whose code is sent along the lower set of bus wires. In order to ensure that the destination can decode data without any ambiguity, the upper segments and the lower segments do not have any overlapping bus wires for code transmission. To ensure the integrity of the codec's operation, all upper segments use the same code width. Likewise, all the lower segments use the same code-width.

Figure 2 shows the encoder algorithm for the TUBE encoder. Since the encoder encodes values of varying widths, the code width is kept constant. For every incoming value, the encoder searches its segments to see if the data value or its bit positions where encountered before. In the event of a hit, the encoder sends the corresponding code along the upper bus wires and raises the external control signal. During a hit in multiple segments, the code from the segment with largest number of bit positions is chosen by

TUBE Decoder Algorithm

```
For each incoming value
Do
        If encode_signal == 0 then
                interpret data as-is
        else
                search upper segments
                decoded_data = upper_hit(data) OR lower (bus_value)
                if internal control signal == 1 then
                        decoded_data = upper_hit(data) OR lower_hit(data)
                end if
        end if
Done
```

Figure 3 shows the TUBE Decoder Algorithm.

the selection logic. This selected code is designated as the upper code. The encoder also searches its lower segments to see if the bit positions were encountered in the recent past.

During a hit in the lower segment, the hit code (lower_code) forms the lower order bits of the encoded bus value. The encoder raises the internal control signal to let the destination know that a code is being sent in both the upper and lower set of bus wires. During a lower segment miss, the lower order bits of the incoming value constitute the lower portion of the encoded bus value. When the encoder does not find a match in any of its segments, the encoder lowers the external control signal and sends the value unencoded.

Figure 3 shows the decoder algorithm for the TUBE decoder. When the external control signal is low, the data is interpreted as-is by the decoder. When the external control signal is high, the decoder searches the upper segments. The decoder searches the lower segments when the value of the internal control signal is set to 1. When the decoder encounters a hit, the data at the hit location is used to obtain the decoded data value

3. EXPERIMENTS

3.1 Experimental setup

We modified the *sim-outorder* simulator in the Simplescalar toolset [2] to incorporate our bus encoding techniques. 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 applications. Our test programs consisted of 16 benchmarks from the MediaBench [6], NetBench [9] benchmark suites and four applications from SPECINT2000 [12].

We modeled two different architectures – an embedded system-like architecture with a small L1 cache and a desktop like architecture with both L1 and L2 cache sizes. For the embedded system like architecture, we evaluated data caches of the following sizes – 1KB, 2KB, 4KB and 8KB. For the desktop-like architecture, we fixed the L1 and L2 cache sizes at 64KB and 512KB respectively. We used the desktop like architecture while simulating the SPECINT applications and simulated the remaining benchmarks using an embedded system like architecture. For each of these cache configurations, we fixed the block size of the instruction and data caches at 32 bytes. Both instruction and data caches have on chip and off-chip latencies of 1 cycle and 100 cycles respectively. The off-chip data bus is 32-bits wide. The off-chip data trace consisted of both instruction and data values.

3.2 Bus power model

We use a bus power model similar to the one discussed by Catthoor et. al [3]. In general estimating the energy used in the off-

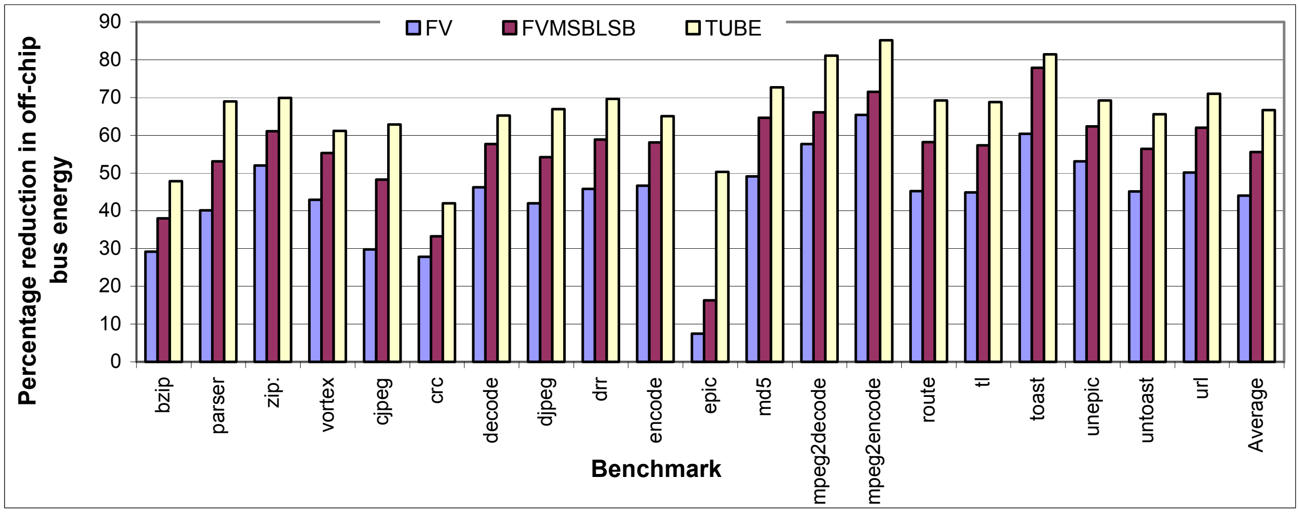

Figure 4 shows the percentage reduction in off-chip energy for different data bus encoding schemes. Cache size for SPECINT : L1=64KB, L2=512KB; Cache size for other applications : L1=2KB, no L2. On an average, TUBE provides an improvement of 21% over FVE and 10%over FVMSBLSB.

chip interconnects is difficult. We can approximate the capacitance for the bus using the formula:

$$C_{bus} = C_{metal} * \textit{Number of Bus lines}$$

In this expression C_{metal} is the capacitance of the metal interconnect for each bus line. Using the numbers given in [3], it is estimated to be 20pF. C_{bus} gives the effective capacitive load to be driven during a bus transaction. We calculated the total bus energy per cycle using the following formula:

$$E_{total} = E_{encoder} + \frac{T_r * C_L * V^2}{\#\ \text{of cycles}} + E_{decoder}$$

where, T_r = *total number of transitions in the off-chip bus*
C_L = *Load capacitance of the off-chip bus line.*
V = *Supply voltage.*

3.3 Codec's Energy

Our codec's hardware design mainly consists of Content Addressable Memories (CAMs), pipelined registers, 160 x 32 MUX and selection logic. By scaling the results in [10] to 0.18µm technology, we find the area, delay and energy of the CAM cell to be 95.68pJ 5ns and 0.045 mm^2 respectively. We model the 32-bit pipelined register using Cacti[11] and we find that the maximum delay, energy consumption, and area requirement is 0.173 ns, 0.443 pJ and 0.00164 mm^2 in 0.18µm technology respectively. We model the 160 x 32 MUX using cacti and found that it requires 0.002120 mm^2 and maximum energy consumption and delay is found to be 0.78 ns and 4.3 pJ respectively. We use the *cadence* tools layout results for primitive gates and estimate the area, maximum energy consumption and delay of selection logic block to be 0.000121 mm^2, 0.354 pJ and 0.16 ns respectively. We calculate the clock frequency based on the CAM delay and it is found to be ~175 MHz. The energy consumption for the codec is found to be 103.12 pJ and the total area requirement of our codec design is found to be 0.0521 mm^2.

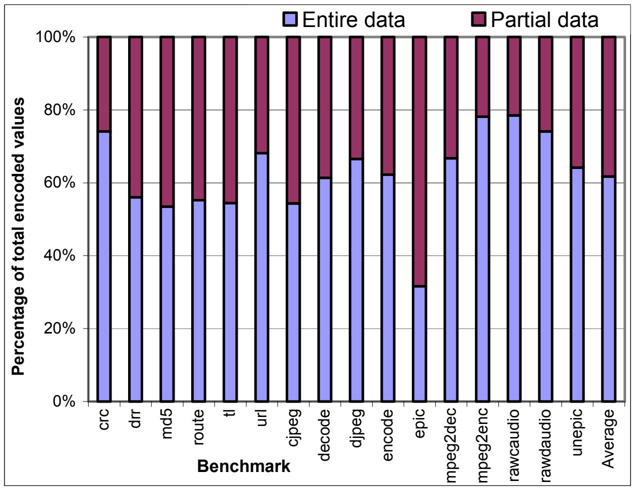

Figure 5 shows the percentage of Entire data Vs partial data amongst encoded bus values. On an average, TUBE manages to encode all the bit positions in nearly 61% of the encoded values.

4. RESULTS

We evaluate our proposed scheme for the configuration described in section 3. Figure 4 shows the percentage reduction in energy while using a L1 cache of size 2KB. We evaluate the energy savings by taking care of energy consumption of hardware codec. For applications like *cjpeg*, TUBE outperforms FVE by 28%. *Epic* application has a lot of hits in the partial segments (MSB and silent) and hence yields significant energy improvement. We observe a similar energy saving trend in spec2000 applications, where *parser* provides an extra energy saving of 18% over FVMSBLSB scheme. However, for *crc* application, energy saving is not significantly high, which demonstrates less data value locality in *crc* program. On an average, TUBE provides an energy improvement of 21% of FVE and 10% over FVMSBLSB, and 64% over unencoded data.

Figure 5 shows the percentage of cases during which the entire and partial data are encoded upon a table hit. As shown in the graph, on nearly 62% of the encoded cases, TUBE codec encodes either a 32-bit contiguous value or it encodes a MSBLSB hit or a silent hot hit. The remaining 39% of the cases consists of MSB or silent segment hits alone. Since the codec manages to encode entire

Table 1. Percentage average energy reduction for different cache configurations

Cache size	FVE	FVMSBLSB	TUBE
1KB	44.94	55.52	66.71
2KB	42.88	53.78	63.89
4KB	40.87	51.06	61.84
8KB	40.62	50.93	61.03

data on 62% of all the encoded values, it yields significant energy savings.

We run our embedded applications for different cache sizes to see how different cache sizes affect the overall energy savings. Table 1 shows the percentage reduction in energy for different cache configurations. We find that there is a slight decrease in the percentage of energy saving with the increasing cache size. The energy saving for 2 KB cache size over FVE is 21%, whereas it reduces to 20.4% for 8 KB cache. This finding can be attributed to the fact that there is a decrease in the off-chip data value locality with increasing cache sizes.

5. RELATED WORK

Frequent Value Encoding (FVE) [15] is a data bus encoding scheme capable of encoding entire data values. FVMSBLSB [13] stores the entire data value, Most Significant Bit (MSB) portions and Least significant Bit (LSB) portions of values in separate tables. The codec sends a one-hot code for subsequent occurrences of the data value or its MSB or LSB portions. While encoding MSB or LSB portions alone, the remaining portion of the data is sent unencoded. Basu et al. [1] proposed a value cache at both ends of the communication channel. During a hit, the index to the cache entry is sent instead of the whole word. Bus Expander [4] and Dynamic Base Register Caching (DBRC) [5] propose compaction techniques to increase the effective bus-width. DBRC uses dynamically allocated base registers to cache the higher order bits of address values. Working Zone Encoding (WZE) [10] 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. However, these schemes fail to exploit locality in non-contiguous bit positions.

Lv et al. [7] 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. This scheme fails to exploit the occurrences of entire data values and consequently, the reduction in switching activity is not significantly high. Wen et. al [14] investigate the use of value prediction techniques to reduce transition activity on the data buses.

TUBE differs from all of the aforementioned works in the following aspect. TUBE captures chunks of varying widths from data values. These chunks can consist of bits from contiguous and non-contiguous bit positions of the data value. For subsequent occurrences of these bit positions, TUBE sends a code instead of data. Control signals require the availability of a free pin on the chip and are hence, very expensive to provide. TUBE uses just one external control signal to indicate the presence of encoded values on the data bus.

6. CONCLUSION

We proposed and evaluated TUBE, a novel bus-encoding scheme for reducing the power consumption in off-chip data buses. TUBE is capable of exploiting repetition in contiguous as well as non-contiguous bit positions in order to encode data values. TUBE requires just one external control signal to encode data.

We performed simulations for 20 workloads covering the SPECINT2000 [12] and commonly used embedded system applications like the Mediabench [6] and Netbench [9]. On an average, TUBE architecture reduces the off-chip bus energy by 64%, which is a 21% improvement over FVE [15]. TUBE achieves as much as 84% reduction in energy over unencoded data for some embedded system applications. We found that TUBE codec can be implemented with a minimal area of 0.0521 mm^2 and has a performance overhead as low as 0.45% of the total execution time. Hence, TUBE design is feasible in terms of energy, performance and delay overhead.

7. REFERENCES

[1] K. Basu, A. Choudhary, J. Pisharath and M. Kandemir, “Power protocol: Reducing Power Dissipation on Off-Chip Data Buses”, *35th Annl IEEE/ACM Symp. on Micro architecture (MICRO-35)* , Istanbul, Turkey, Nov. 2002.

[2] D. Burger and T. Austin, “The SimpleScalar Tool Set, Version 2.0, Technical Report”, University of Wisconsin-Madison, Computer Science Department, 1997

[3] F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele and A. Vandecappelle, “ Exploration of Memory Organization for Embedded Multimedia System Design”, *Kluwer Academic Publishers”,* 1998.

[4] D. Citron and L. Rudolph, “Creating a Wider Bus using Caching Techniques”, *Proceedings of the first International Symposium on High Performance Computer Architecture*, pp 90-99, January 1995.

[5] M. Farrens and A. Park, “ Dynamic Base Register Caching: A technique for Reducing Address Bus width”, In Proceedings of 18th International Symposium on Computer Architecture (ISCA), pp 128-137, Toronto, Canada, May 1991.

[6] C. Lee, M. Potkonjak and W. Mangione-Smith, “MediaBench: a tool for evaluating and synthesizing multimedia and communications systems”, *Intl. Symp. on Microarchitecture*, pages 330-335, 1997.

[7] T. Lv, J. Henkel, H. Lekatsas and W. Wolf, “An Adaptive Dictionary Encoding Scheme for SOC Data Buses”, *Design Automation and Test in Europe,* Paris, France, Mar. 2002.

[8] I.Y.L. Hsiao, D.H. Wang, and C.W. Jen, “Power modeling and low-power design of content addressable memories”. *In Proceedings of the IEEE International Symposium on Circuits and Systems*, volume 4, pages 926-929, 2001.

[9] G. Memik, W. H. Mangione Smith and W. Hu, “NetBench : A Benchmarking Suite for Network Processors”, *Intl. Conf. on Computer Aided Design (ICCAD)*, pp 39-42,San Jose, California, Nov. 2001

[10] E. Musoll, T. Lang and J. Cortadella, “Working Zone Encoding for Reducing the Energy in Microprocessor Address Buses”, *IEEE Trans. on VLSI Systems*, pp 568-572, Volume 6, Dec. 1998.

[11] P.Shivakumar and N. P. Jouppi, “Cacti 3.0: An Integrated Cache Timing, Power and Area Model," *Western Research Lab (WRL) Research Report 2001.*

[12] SPECINT2000, http://www.specbenh.org/cpu2000

[13] D. C. Suresh, B. Agrawal, J. Yang, W. Najjar and L. Bhuyan, “Power Efficient Encoding Techniques for Off-Chip Data Buses”, In the Proc. of *Compilers and Architecture and Synthesis for Embedded Systems (CASES),* San Jose, CA, Oct. 2003

[14] V. Wen, M. Whitney, Y. Patel and J. D. Kubiatowicz, “Exploiting Prediction to Reduce Power on Buses”, *In Proc. of the 10th Intl. Symp. on High Performance Computer Architecture*, pages 2-13, Madrid, Spain, Feb 2004.

[15] J. Yang and R. Gupta, “FV-Encoding for Low Power Data I/O”, *ACM/IEEE Intl. Symp. on Low Power Electronic Design*”, Pages 84-87, 2001.