Context-independent Codes for Off-chip Interconnects

Kartik Mohanram and Scott Rixner

Rice University, Houston TX 77005, USA
kmram@rice.edu and rixner@rice.edu

Abstract. This paper introduces the concept of context-independent coding using frequency-based mapping schemes in order to reduce off-chip interconnect power consumption. State-of-the-art context-dependent, double-ended codes for processor-SDRAM off-chip interfaces require the transmitter and receiver (memory controller and SDRAM) to collaborate using the current and previously transmitted values to encode and decode data. In contrast, the memory controller can use a context-independent code to encode data stored in SDRAM and subsequently decode that data when it is retrieved, allowing the use of commodity memories. In this paper, a single-ended, context-independent code is realized by assigning limited-weight codes using a frequency-based mapping technique. Experimental results show that such a code can reduce the power consumption of an uncoded off-chip interconnect by an average of 30% with less than a 0.1% degradation in performance.

1 Introduction

Modern embedded networking, video, and image processing systems are typically implemented as systems-on-a-chip (SoC) in order to reduce manufacturing costs and overall power and energy consumption. By integrating all of the peripheral functionality directly onto the same chip with the core microprocessor, both chip manufacturing and system integration costs can be lowered dramatically. In addition to cost, managing power and energy is a first order constraint that drives the design of embedded systems based on SoCs. However, most modern SoC-based embedded systems require more memory capacity than can reasonably be embedded into a single core. In such systems, the interconnect between the processor and external memory can consume as much or more power than the core itself. Even though external memory and its associated interconnect are major contributors to the overall power dissipation in SoC-based embedded systems, such systems will continue to require the memory capacity afforded by external memory into the foreseeable future. Therefore, it is essential to develop advanced memory controller architectures to reduce the power dissipation of external memories and the interconnect between the embedded SoC core and that memory.

Encoding data that is stored in memory can minimize the power consumed by the processor-memory interconnect. Dynamic power is consumed by the interconnect drivers when there are bit transitions. To minimize this power, double-ended, context-dependent codes such as the bus-invert code have previously been proposed. Double-ended codes encode data at the transmitter and decode it at the receiver. For a processor-memory interconnect, this implies that the SDRAM also needs to participate in such

codes. Context-dependent codes use the value last transmitted on the interconnect as well as the current data in order to encode the data to minimize transitions. For example, bus-invert coding either transmits the data value unchanged or its inverse, depending on which value minimizes transitions on the interconnect. If the SDRAM were modified to support such coding, bus-invert coding could reduce transitions on the interconnect by 22% on average.

In contrast, single-ended, context-independent codes are much simpler to implement, as they do not require modifications to the SDRAM. This paper introduces the concept of frequency-based, single-ended, context-independent codes for interconnect power reduction. The simplest frequency-based code simply remaps the input space based upon the measured or expected frequency of occurrence of each data value. Despite the fact that such a code is context-independent, and so does not account for possible switching on the interconnect, it is able to reduce the transitions on the interconnect by 28% on average. This simple code results in a larger power decrease on the interconnect than context-dependent bus-invert codes that are explicitly designed to minimize switching activity. Furthermore, frequency-based coding can also be used to augment limited-weight codes (LWCs). A limited-weight code maps the input data to a wider codeword in which the number of ones in the word is restricted. The proposed frequency-based assignment of codewords using a LWC can reduce transitions on the interconnect by an average of 30% over the uncoded case.

Frequency-based context-independent codes reduce interconnect power consumption without requiring the use of specialized SDRAM. Frequency-based codes are effective because frequently occurring values usually follow either themselves or other frequently occurring values on the interconnect. So, if the most frequently occurring values are all mapped to codewords that are close (Hamming distance-based) to each other, then switching activity can be minimized. In this manner frequency-based codes simply, but effectively, reduce dynamic power consumption on interconnects to commodity SDRAM.

The rest of this paper is organized as follows. The following section gives additional background on power dissipation within embedded systems, further motivating the need for new memory controller architectures. Section 3 introduces state-of-the-art coding techniques to reduce power consumption and discusses their limitations. In Section 4, the proposed frequency-based limited-weight codes for low power consumption are described. Section 5 describes a memory controller architecture for these coding techniques. Section 6 describes the experimental setup and the benchmarks used. Section 7 analyzes the performance of the proposed memory controller innovations on this set of embedded computing benchmarks. Finally, Section 8 concludes the paper.

2 Power Dissipation in Embedded Systems

The architecture of a modern SoC-based embedded system is presented in Figure 1. The SoC core has one or more simple processors, designed to provide enough computational capability for the application, integrated with some embedded memory and a variety of on-chip peripherals for data acquisition and connectivity. These systems also integrate SDRAM, since they frequently require more memory capacity—to buffer

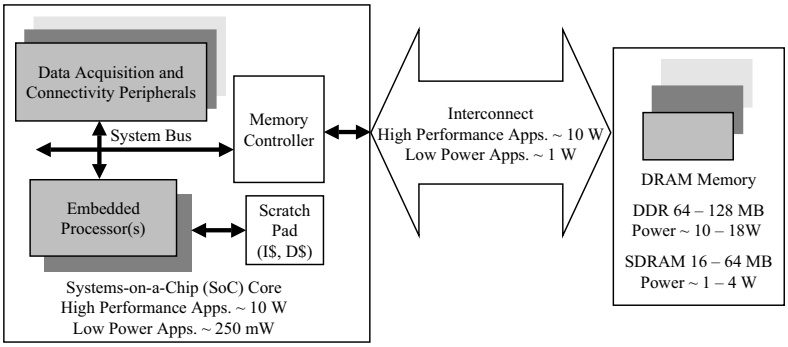

Fig. 1. Typical SoC Embedded System Architecture and Approximate Power Consumption

large data streams before either processing or forwarding them—than can reasonably be embedded into the SoC core.

Managing power dissipation and providing sufficient on-chip memory capacity are two major challenges in the design of such SoC-based embedded systems. The International Technology Roadmap for Semiconductors (ITRS) predicts that without significant architectural and design technology improvements, the power consumption of both high performance and low power SoC-based embedded systems will grow exponentially, easily exceeding power budgets [1, 2]. Tethered embedded systems frequently have limited power budgets because of constraints on power delivery and cooling area available on peripheral buses. Mobile systems, in addition to requiring low power dissipation, are also constrained by battery life making energy consumption an important factor.

The annotations in Figure 1 show that, currently, the power dissipation of representative low power and high performance embedded systems is divided roughly equally among the SoC, the memory interconnect, and the external memory. Furthermore, while high performance embedded systems can dissipate an order of magnitude more power than low power systems, the relative power dissipation of the SoC core, the interconnect, and the memory remains similar. It is clear that in such systems, the external memory and interconnect can dissipate as much or more power than the SoC core. Thus, the memory system and the interconnect are candidates for techniques to reduce and manage power and energy.

Dynamic power is dissipated on a signal line of a bus whenever there is a transition on that line. A signal transition causes the drivers to actively change the value of the bus, which acts as a large capacitance. It is also possible for the drivers to dissipate static power when they hold the bus at either logical 0 or logical 1, depending on the design of the drivers. It is possible to limit this leakage power for the low frequencies of operation commonly found in embedded systems, so static power dissipation is typically dwarfed by the dynamic power dissipation of the bus drivers. However, there are still situations in which static power dissipation cannot be ignored, including higher frequencies of operation and when there are voltage mismatches between the core and the memory.

3 Related Work: Coding to Reduce Power

The techniques used to reduce power dissipation in external memory systems fall roughly into three categories: low-power memory modes, external memory access reduction, and double-ended techniques. Most modern commodity memories have one or more low power modes of operation. It may be expensive to enter and exit these modes, but frequently the memory dissipates an order of magnitude less power when it is in these modes. Several techniques, such as those proposed in [3] and [4], can be used to determine when external memory should be powered down to minimize power dissipation without disrupting performance. Another way to reduce the power dissipation of external memories is to access them less frequently. These techniques use some combination of on-chip memory, caching, and code reorganization to allow the processing core to reduce the number of external memory accesses [5–8]. In turn, this reduces the power demands of the external memory when it is active and can also allow it to be put to sleep more frequently. The final set of techniques for reducing the power dissipation of external memories require cooperation between the memory controller and the memory. These techniques either encode data to minimize power dissipation across the interconnect or transmit additional information to the memory to enable it to access the memory array more efficiently [9–13].

The majority of data encoding schemes proposed in literature are not applicable to the off-chip interconnect between an SoC and external memory because they are double-ended, context-dependent codes. Double-ended codes require collaboration between the transmitter and receiver to transfer encoded data. In such state-of-the-art codes, the transmitter (i.e., the memory controller on the SoC) uses a potentially complex handshaking protocol to communicate with the receiver (i.e., a decoder in the memory), which has the ability to interpret these handshakes to decode the transmitted data. The roles of the coder and the decoder would be reversed when communicating in the opposite direction (i.e., a memory read). So, a potentially complex codec (coder-decoder) has to be present on both ends to successfully use these schemes. However, commodity SDRAMs do not have a built-in codec that is capable of communicating with the SoC core in this fashion.

Context-dependent coding schemes rely on inter-symbol correlation on successive data transfers to reduce power consumption. However, such schemes are not effective with commodity memory, as the memory cannot participate in the scheme. Therefore, any coding scheme using commodity memory must be able to unambiguously decode data read from the memory that was encoded when it was written to the memory. If inter-symbol correlation information is used when writing the data, then that information is not available upon reading the data, since there is no guarantee that data will be read in exactly the same order it is written. Some context-dependent coding schemes, such as those that use an XOR decorrelator, do not include enough information in the codeword to unambiguously recover the original data without the context information. However, other context-dependent schemes, such as bus-invert coding, produce codewords that can be decoded without context information. Even then, such schemes will only minimize power when writing to the memory, as the data will be read in a different context than it was written. Therefore, context-dependent codes are almost exclusively used in situations where both the transmitter and receiver can participate in the code.

That way, the context information can be used to decode the transferred data before it is stored. If the data is retrieved later, it is re-encoded and transferred based on the context at that time.

The most popular and easy-to-implement double-ended context-dependent code reported in literature in the bus-invert code [14]. The bus-invert code is a context-dependent, double-ended code since it computes the Hamming distance between the currently encoded data value on the bus and the next data value. If the Hamming distance exceeds $\lceil \frac{n}{2} \rceil$, then the transmitter inverts the next value transmitted on the bus. An additional line on the bus indicates whether the data is inverted or not, allowing the receiver to unambiguously decode the transmitted data. In this manner, an n-bit value can be transmitted over an $n+1$-bit bus with at most $\lfloor \frac{n+1}{2} \rfloor$ transitions. Without such coding, an n-bit value could cause as many as n transitions over an n-bit bus. For example, if the current value on the bus is 0000, and the next value to be transfered is 0001, then the Hamming distance between the values is 1. Therefore, 0001 will be transmitted over the bus with the invert bit set to 0, indicating the data is not inverted. However, if the current value on the bus is 1111 instead, the Hamming distance between the values is 3 and hence 1110 is transmitted with the invert bit set to 1. In this manner, each information symbol in the n-bit input space maps to two codewords. The codeword that minimizes switching activity on the interconnect is chosen for transmission to reduce power consumption. Bus-invert is thus not a one-to-one mapping, i.e., it is not a context-independent code. The bus-invert codewords for all the information symbols on a 4-bit wide data bus are shown in column 2 of Table 1.

Many other context-dependent, double-ended codes have been proposed. One such code is based on the use of a decorrelator, which XOR's the data to be transmitted with the previous value transmitted across the bus [15, 16]. The receiver must then recover the actual value by undoing the XOR operation. Further reductions can be achieved by exploiting information about the frequency of occurrence of particular data values on the bus. In [17], a decorrelator was combined with a one-hot encoding of the 32 most frequently occurring values. Like bus-invert, such frequent value encoding is still a context-dependent, double-ended code because of the use of the decorrelator. The transmitter first decides if the data value is one of the most 32 frequently occurring values. If so, it is one-hot encoded. A one-hot code on a n-bit wide bus is a coding scheme where exactly one out of n bits is set to one. At the word-level, 32 codewords are available and hence 32 frequently occurring values can be encoded leaving the remaining values unencoded.** Note that an additional bit is needed to indicate whether or not the data is one-hot encoded to the receiver. The result of one-hot encoding is then passed through the decorrelator prior to transmission across the bus. The receiver must recover the actual value by undoing the XOR and one-hot encoding transformations. The final column of Table 1 shows the one-hot codeword assignments, based on the frequency distribution in column 4, for frequent value coding on a 4-bit wide data bus. Note that only 4 most frequently occurring values (1101, 1001, 0111, 0100) are one-hot encoded. In practice, these values would also be XOR'ed with the previous value transmitted

** The reported code also ignored values 1-16 and performed equality tests before transmission, the details of which are excluded for brevity [17]. Nevertheless, our experimental setup implemented the best scheme reported in [17] that includes some of these features.

Table 1. Comparison of Different Codes

Information Symbols	Bus-invert Coding [14]	2-LWC Code [13]	Frequency (%)	Freq.-based remapping	Freq.-based 2-LWC	Freq. value coding [17]
0000	0 0000 1 1111	0 0000	6.7	1001	0 0110	0 0000
0001	0 0001 1 1110	0 0001	5.6	0101	0 1010	0 0001
0010	0 0010 1 1101	0 0010	4.7	1011	1 1000	0 0010
0011	0 0011 1 1100	0 0011	6.9	1010	0 0011	0 0011
0100	0 0100 1 1011	0 0100	7.6	0100	0 0100	1 1000
0101	0 0101 1 1010	0 0101	7.0	1100	1 0000	0 0101
0110	0 0110 1 1001	0 0110	4.0	1101	1 0010	0 0110
0111	0 0111 1 1000	1 1000	8.1	0010	0 0010	1 0100
1000	0 1000 1 0111	0 1000	4.8	1110	0 1001	0 1000
1001	0 1001 1 0110	0 1001	8.4	0001	0 0001	1 0010
1010	0 1010 1 0101	0 1010	5.9	0011	0 1100	0 1010
1011	0 1011 1 0100	1 0100	4.0	0111	1 0100	0 1011
1100	0 1100 1 0011	0 1100	6.6	0110	0 0101	0 1100
1101	0 1101 1 0010	1 0010	8.5	0000	0 0000	1 0001
1110	0 1110 1 0001	1 0001	3.7	1111	1 0001	0 1110
1111	0 1111 1 0000	1 0000	7.5	1000	0 1000	0 1111

across the bus by the decorrelator. Like bus-invert, frequent value encoding does not use a one-to-one mapping, as a particular data value can map to many encoded values depending on the previous data transmitted across the bus.

4 Context-independent, Single-ended Codes

This section introduces and develops a class of context-independent, single-ended coding schemes for embedded applications. These coding schemes are split into two phases.

During the first phase, a set of codewords is generated. In the second phase, each information symbol is assigned to a unique codeword. These assignments are determined purely by frequency-based metrics without using any local context information. Such codes have several advantages. First, they significantly lower power consumption on the interconnect between the SoC and the memory modules. Second, they are single-ended i.e., they do not require the SDRAM to participate in the coding-decoding process. A codec is only required in the memory controller on the SoC. Last, they have negligible impact on performance during the coding-decoding process.

4.1 Limited-weight Codes (LWCs)

One class of context independent codes that meet all the above requirements are limited-weight codes (LWCs) [13]. Consider a k-bit wide data bus with 2^k information symbols. A m-LWC is a one-to-one mapping where every word in the 2^k input space maps to a codeword such that the Hamming weight (i.e., the number of ones in the codeword) is less than or equal to m. Since the source entropy must remain unchanged, i.e., since every information symbol must have a unique codeword, the following inequality must be satisfied by all m-LWCs:

$$\binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{m} \geq 2^k \tag{1}$$

Here, n is the minimum number of bits ($m \leq n$) such that the inequality is satisfied, i.e, n determines the width of the bus needed to implement a m-LWC. Note that the inequality is only satisfied when $n \geq k$. A perfect m-LWC satisfies Equation 1 above with equality, i.e., all the codewords of length n with weight less than or equal to m are used in the mapping. For example, a 4-LWC where k equals 8 is a perfect 4-LWC when the codeword bus width n equals 9. The information symbols and the corresponding codewords for the perfect 2-LWC when k equals 4, obtained using the generation technique presented in [13], are presented in columns 1 and 3 respectively in Table 1.

4.2 Frequency-based Codes

Frequency-based codes are a class of codes where a context-independent mapping between information symbols and codewords is achieved by assigning information symbols that have the highest probability of occurrence to codewords with minimum weight. As discussed in Section 3, one way to use frequency would be to one-hot encode a small set of frequently occurring values at the word-level to achieve power savings. However, such an approach does not use the codeword space efficiently since only 32 out of 2^{32} values can actually be encoded by this scheme.

A more efficient way to use frequency is to remap information symbols to be transmitted on the n-bit wide bus to codewords on a n-bit wide bus, i.e., through a permutation. Such a remapping can be statically determined by an analysis of several traces in the application space. The frequency distribution of information symbols from each application could be used for that application, or the frequency distributions for a set of applications could be combined to produce a global frequency distribution. The information symbols are then ranked in descending order of frequency of occurrence,

and remapped to codewords in increasing order of weight. Thus, the information symbol that occurs most frequently on the bus is remapped to the codeword with the least weight. In practice, such a remapping would have to occur at the byte-level since word-level remapping is impractical.

The frequency distribution given in Table 1 can be used to perform such a remapping of the information symbols in the table. The information symbols and the corresponding codewords for a frequency-based remapping of the 4-bit information space are presented in columns 1 and 5 respectively in Table 1. The frequency distribution used to generate this remapping is presented in column 4 in the same table. For example, the most frequently occurring information symbol, 1101, would be remapped to 0000.

4.3 Frequency-based m-LWCs

The biggest handicap of m-LWCs is that the one-to-one mapping is statically determined without any knowledge of the characteristics of the input space. The main contribution of this paper is to combine the advantages of frequency-based codes with LWCs to produce a context-independent single-ended mapping. By combining m-LWCs with frequency-based coding, the distribution of information symbols is analyzed to produce a context-independent mapping that, while statically determined, exploits an a priori knowledge of the distribution of information symbols. Frequency-based m-LWCs thus leverage the advantages of both frequency-based coding and limited-weight coding. It is a departure from conventional types of codes that seek to explicitly minimize transitions using the state of the bus. The generation of the codewords is separated from the mapping process, and the best of both techniques is harnessed to realize practical context-independent single-ended codes.

The frequency-based mapping encodes information symbols with the highest frequency of occurrence to LWC codewords with the least weight. A simple frequency-based mapping from a 2-LWC to a 4-bit information space, using the frequency distribution in column 4 of Table 1 is presented in column 6 of the same table.

5 Memory Controller Architecture

Figure 2 shows the architecture of a memory controller for embedded systems. As the figure shows, memory requests arrive on the system bus. At this point, if the memory request is a write, the data will be encoded by the context-independent encoder before it is placed in a queue within the memory controller. The SDRAM controller within the memory controller then issues the appropriate commands to the SDRAM in order to satisfy each pending request in the queue. Finally, if the memory operation is a read from the SDRAM, the data can be decoded by the context-independent decoder before being returned to the core over the system bus.

A context-independent codec does not need to be near the pins. Rather, the data can be encoded and decoded anywhere within the memory controller because only the actual data being encoded or decoded is needed to perform the encoding or decoding. This makes it convenient to encode write data before it is placed in the memory queue,

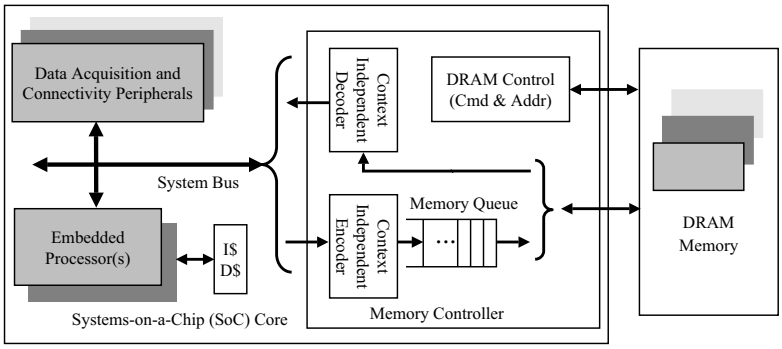

Fig. 2. Memory controller architecture for embedded systems

thereby minimizing any latency penalties. It is entirely possible that the latency of encoding write data can be hidden by long latency SDRAM operations that must occur before the data can cross the pins anyway. Similarly, read data can be decoded as it is sent to the system bus. Again, the decoding latency could possibly be hidden by arbitration delays for the system bus. For the 30 benchmark programs from the MiBench suite that will be explored in Section 7, an extra cycle latency penalty for decoding results in less than a 0.1% performance penalty on average.

A context-independent codec can be implemented in multiple ways. In the most general case, such as frequency-based coding, a lookup-table is the most efficient mechanism. To encode or decode bytes, a 256-entry table would be required with either 8 or 9 bit entries, depending on the code. For performance, it is likely that multiple identical tables would be required, one for each byte that can be transferred on the system bus in a given cycle. If the code is fixed, then the lookup tables can be compact ROM tables. To provide the flexibility to change the code, however, it is likely that the lookup-tables would have to be SRAM structures. Combinational logic can be used to implement more regular context-independent codes. For example, the limited weight code described in [13] could be implemented using a simple population count and possible inversion.

Finally, many of the codes discussed here increase the size of the data by adding an additional bit for every byte. This would increase the datapath width of the memory controller, the width of the processor-memory interconnect, and the width of the SDRAM. Obviously, this additional bit can increase power consumption, but the objective of these codes is to reduce power consumption by limiting the number of transitions, so usually this is not an issue in the memory controller or the processor-memory interconnect, as will be shown in Section 7 (all results include the transitions on this additional wire, as appropriate). However, widening the SDRAM is potentially problematic. Many SRAMs designed for embedded systems have 9-bit bytes. And Samsung is starting to introduce SDRAMs of that nature, as well [18]. The wider Samsung SDRAMs consume 6–8% more current than their normal counterparts. However, this is assuming a regular data pattern. In practice, the reduction in switching activity achieved by these codes can more than offset this increase.

Table 2. Average reduction in transitions.

Code		Reduction (%)
Context-dependent Double-ended	Bus Invert	21.8
	Self FV32 with Decorrelator	38.7
Context-independent Single-ended	Self FV32	17.8
	Self FV8	15.5
	4-LWC	13.9
	Self 8-LWC	28.2
	Global 8-LWC	22.4
	Self 4-LWC	30.3
	Global 4-LWC	25.1

6 Simulation Infrastructure

The coding techniques presented here were evaluated using the SimpleScalar/ARM simulator [19]. The simulator was configured to closely match the Intel Xscale processor [20]. The Xscale can fetch a single instruction per cycle and issues instructions in order. Branches are predicted with a 128 entry branch target buffer and a bimodal predictor. The instruction and data caches are each 32 KB and have a single cycle access latency. The caches are configured as 32-way set associative and use a round-robin replacement policy. SimpleScalar was also modified to incorporate a cycle accurate SDRAM model so that all SDRAM accesses occur as they would in an actual system.

The SDRAM simulator accurately models the behavior of the memory controller and the SDRAM. The SDRAM model simulates all timing parameters, including command latencies, all required delays between particular commands, and refresh intervals. The memory controller within the simulator obeys all of these timing constraints when selecting commands to send to the SDRAM, thereby accurately representing the sequence of data transferred over the processor-memory interconnect. The simulator is configured to model a 75 MHz, 512 Mb Micron MT48LC32M16A2-75 single data rate SDRAM [21].

The bit transitions on the interconnect for the encoded and unencoded data transfers was calculated as the SDRAM is accessed. This faithfully models the bit transitions that would occur on the data bus in the appropriate order.

The MiBench embedded benchmark suite was used to evaluate the proposed coding techniques [22]. Thirty applications are used from the suite with their large input sets. While still small, the large inputs are more representative of actual workloads. The applications span the automotive, consumer, networking, office, security, and telecomm domains.

7 Results

Table 2 shows the average reduction in transitions on the processor-memory interconnect for nine coding strategies when compared with the baseline uncoded case. The

first two codes in the table are context-dependent, double-ended codes. As described in Section 3, bus invert coding is the simplest and most popular such code, and FV32 with a decorrelator one-hot encodes the 32 most frequently occurring values (for each benchmark) and uses a decorrelator to significantly reduce switching activity. As the table shows, both context-dependent, double-ended codes perform quite well, reducing transitions on the interconnect by 21.8% and 38.7%, respectively.

The remaining seven codes are all context-independent, single-ended codes that can be implemented entirely within the memory controller without specialized SDRAM. FV32 and FV8 simply one-hot encode the 32 most frequently occurring word values or the eight most frequently occurring byte values to form a code as presented in Table 1. These codes are labeled "self", as each benchmark uses the most frequently occurring values from that benchmark. As the table shows, these codes perform poorly, yielding only a 17.8% and 15.5% reduction in switching activity. Therefore, such a one-hot encoding strategy relies heavily on a context-dependent, double-ended decorrelator to reduce transitions on the interconnect.

4-LWC is the original limited-weight code, presented in Section 4, which uses nine bits per byte to encode all byte values with at most four bits set. This code reduces switching activity by 13.9% on average.

The final four limited-weight codes use the frequency-based assignment scheme presented in this paper. "Self" and "Global" refer to whether each benchmark's own frequency distributions were used to assign codewords for that benchmark or all benchmarks used the same codewords derived from the frequency distributions of all benchmarks. The 8-LWC codes use eight bits to encode each byte, with up to eight bits set, yielding a simple remapping. The 4-LWC codes again encode each byte using nine bits with at most four bits set. As the table shows, these codes are able to reduce transitions on the interconnect by 22.4–30.3% on average. As would be expected, the codes which use the frequency distributions for each benchmark individually yield higher reductions, by about 5–6%. These results show that when using limited-weight codes, the assignment strategy is critical. Furthermore, the penalty of using an extra wire for the 4-LWC codes is more than offset by the effectiveness of such codes (the results include the switching on the additional wire).

8 Conclusions

State-of-the-art coding techniques to reduce power consumption across the processor-memory interconnect have traditionally used context-dependent, double-ended techniques. This requires specialized memory that can participate in such codes. This paper introduced viable context-independent, single-ended codes that are competitive with these state-of-the-art codes, but can be used with commodity memory.

The proposed codes are effective at reducing power without degrading performance for thirty applications from the MiBench embedded benchmark suite with their large input sets. Frequency-based remapping codes require no augmentation to the memory bus or modules. These codes reduce the transitions of the uncoded data stream by 28.2% on average, and minimally impact performance (by less than 0.1% on average) across the set of thirty benchmark programs. Frequency-based 4-LWCs reduce the transitions

of the uncoded data stream by 30.3% on average, improve on context-dependent double-ended bus-invert codes by 10.9% on average, and minimally impact performance (by less than 0.1% on average) across the set of thirty benchmark programs.

This paper has shown that both the type of code used and the assignment scheme for that code are important. Limited-weight codes by themselves are ineffective. Similarly, using frequency information without limited-weight codes yields an inefficient code that is also ineffective. However, using frequency information to assign limited-weight codes minimizes transitions to a greater extent than any other context-independent, single-ended code. Furthermore, such codes sometimes outperform context-dependent, double-ended codes that cannot be used with commodity SDRAMs. Since embedded systems continue to use commodity memories and the processor-memory interconnect is a dominant consumer of power in such systems, the coding techniques presented here can significantly improve the overall power efficiency of modern embedded systems.

References

1. Edenfeld, D., Khang, A.B., Rodgers, M., Zorian, Y.: 2003 technology roadmap for semiconductors. IEEE Computer (2004)
2. International technology roadmap for semiconductors (2003)
3. Delaluz, V., Kandemir, M., Vijaykrishnan, N., Sivasubramaniam, A., Irwin, M.: DRAM energy management using software and hardware directed power mode control. In: Proceedings of the International Symposium on High-Performance Computer Architecture. (2001)
4. Fan, X., Ellis, C., Lebeck, A.: Memory controller policies for dram power management. In: Proceedings of the International Symposium on Low Power Electronics and Design. (2001)
5. Catthoor, F., Wuytack, S., DeGreef, E., Balasa, F., Nachtergaele, L., Vandecappelle, A.: Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design. Kluwer Academic Publishers (1998)
6. Kulkarni, C., Catthoor, F., DeMan, H.: Code transformations for low power caching in embedded multimedia processors. In: Proceedings of the International Parallel Processing Symposium. (1998)
7. Kulkarni, C., Miranda, M., Ghez, C., Catthoor, F., Man, H.D.: Cache conscious data layout organization for embedded multimedia applications. In: Proceedings of the Design and Test in Europe Conference. (2001)
8. Panda, P.R., Dutt, N.D., Nicolau, A.: On-chip vs. off-chip memory: the data partitioning problem in embedded processor-based systems. ACM Transactions on Design Automation of Electronic Systems **5** (2000) 682–704
9. Benini, L., Macii, A., Poncino, M., Scarsi, R.: Architectures and synthesis algorithms for power-efficient bus interfaces. **19** (2000) 969–980
10. Ramprasad, S., Shanbag, N.R., Hajj, I.N.: A coding framework for low power address and data buses. IEEE Transactions on VLSI Systems **7** (1999) 212–221
11. Sotiriadis, P., Chandrakasan, A.: A bus energy model for deep sub-micron technology. IEEE Transactions on VLSI Systems **10** (2002) 341–350
12. Sotiriadis, P., Tarokh, V., Chandrakasan, A.P.: Energy reduction in VLSI computation modules: An information-theoretic approach. IEEE Transactions on Information Theory **49** (2003) 790–808
13. Stan, M.R., Burleson, W.P.: Low-power encodings for global communication in CMOS VLSI. IEEE Transactions on VLSI Systems (1997)

14. Stan, M.R., Burleson, W.P.: Bus invert coding for low power I/O. IEEE Transactions on VLSI Systems (1995) 49–58
15. Benini, L., DeMicheli, G., Macii, E., Sciuto, D., Silvano, C.: Asymptotic zero-transition activity encoding for address busses in low-power microprocessor-based systems. In: Proceedings of the Great Lakes Symposium on VLSI. (1997) 77–82
16. Musoll, E., Lang, T., Cortadella, J.: Exploiting locality of memory references to reduce the address bus energy. In: Proceedings of the International Symposium on Low Power Electronics Design. (1997) 202–207
17. Yang, J., Gupta, R., Zhang, C.: Frequent value encoding for low power data buses. ACM Trans. Des. Automation Electronic Systems **9** (2004) 354–384
18. Samsung: 256/288 Mbit RDRAM K4R571669D/ K4R881869D data sheet, version 1.4 (2002)
19. Austin, T., Larson, E., Ernst, D.: SimpleScalar: An infrastructure for computer system modeling. IEEE Computer (2002)
20. Clark, L.T., Hoffman, E.J., Miller, J., Biyani, M., Liao, Y., Strazdus, S., Morrow, M., Velarde, K.E., mark A. Yarch: An embedded 32-b microprocessor core for low-power and high-performance applications. IEEE Journal of Solid-state Circuits **36** (2001) 1599–1608
21. Micron: 512Mb: x4, x8, x16 SDRAM MT48LC32M16A2 data sheet (2004)
22. Guthaus, M.R., Ringenberg, J.S., Ernst, D., Austin, T.M., Mudge, T., Brown, R.B.: MiBench: A free, commercially representative embedded benchmark suite. In: IEEE 4th Annual Workshop on Workload Characterization. (2001)