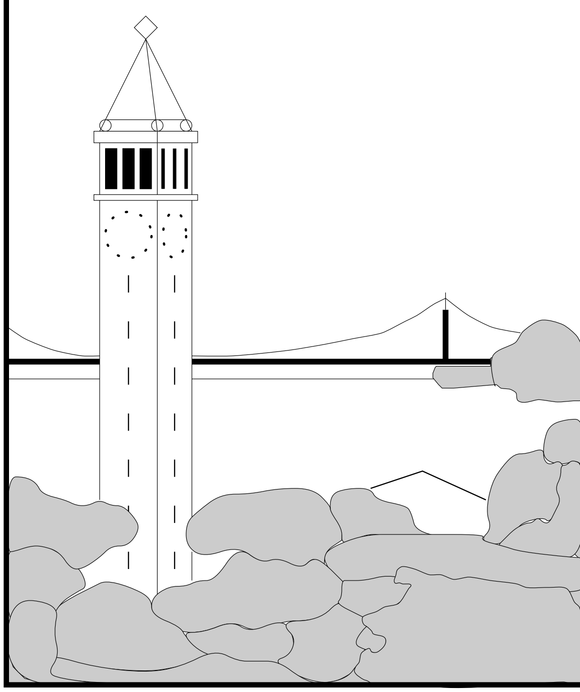
# Exploiting Prediction to Reduce Power on Buses

*Victor Wen*

*vwen@cs.berkeley.edu*



**Report No. UCB/CSD-3-1294**

November 2003

Computer Science Division (EECS)
University of California
Berkeley, California 94720

# Exploiting Prediction to Reduce Power on Buses

Victor Wen
vwen@cs.berkeley.edu

## Abstract

*We explore the possibility of reducing energy consumed by on-chip buses via stateful and stateless coding techniques. We explore the design of a number of simple coding schemes and simulate them using a modified SimpleScalar simulator and SPEC benchmarks. We show an average of 36% savings in transitions on internal buses such as the reorder buffer and register file. To quantify actual power savings, we design a simple dictionary based encoder/decoder circuit in a 0.13$\mu m$ process, extract it as a netlist, and simulate its behavior under SPICE. Utilizing a realistic wire model with repeaters, we show that we can* break even *at median length scales of less than 11.5mm at 0.13$\mu m$ and project a break-even point of 2.7mm for a larger design at 0.07$\mu m$.*

## 1. Introduction

Scaling trends have continually increased the importance of wires relative to logic. Among other things, the ever rising ambitions of computer architects have caused wire lengths to remain constant or increase – even as transistor sizes have shrunk. This observation suggests that energy conscious designers should focus some of their attention on the energy dissipated by on-chip wires. Clearly, this process can involve a variety of techniques at the level of technology, circuits, and architectures. Many of these techniques are complementary.

In this paper, we exploit abundant transistors to transform information into a form that is less energy costly to communicate; our techniques are complementary to other options such reducing voltage swing. Energy is consumed when wires change state. Thus, our goal will be to eliminate or reduce the total number of wire transitions, with a focus on adjacent wires to reduce cross-coupling energy. Compression techniques have long been used to reduce the volume of off-chip communication. Given the large capacitances of cross-chip interconnects, compression circuits can easily save more power than they utilize. In contrast, we address a more difficult question: is it possible to reduce the traffic over *on-chip* buses and *save energy while doing so*?

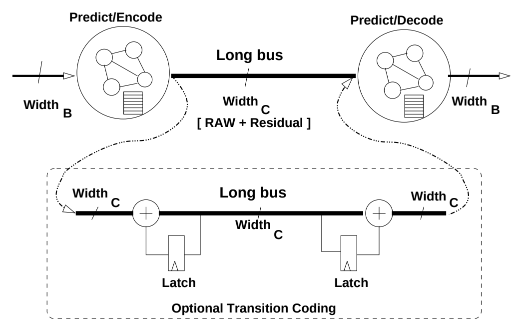



**Figure 1. *Bus transcoding to reduce power:* Arbitrary prediction and/or compression algorithms can be used to reduce traffic on long buses—as long as the state at either end is kept consistent. Optional *transition-coding* can be inserted to reduce the coding problem to one of minimizing Hamming weight of successive values.**

Since on-chip wires have capacitances which are orders-of-magnitude smaller than cross-chip interconnects, the answer to this question requires careful accounting of the energy consumed by the encoding and decoding process.

### 1.1. The Bus Transcoder

Figure 1 shows the basic idea, which we call "bus transcoding" (hereafter called "transcoding"). Circuits at either end of a long bus reduce the number of bus transitions. The encoder takes in $W_B$ bits and encodes them as $W_C$ bits. These bits are then transmitted along the wire (with repeaters as necessary). At the destination, the decoder takes in $W_C$ bits and restores the original $W_B$ bits. In general, $W_B \neq W_C$. For this paper, we will assume that the encoder and decoder are operating synchronously. In this case, these elements can have arbitrarily complicated internal state; it is assumed that the encoder FSM utilizes the input stream to make its state transitions, while the decoder FSM uses the output stream to make its transitions. The goal of this transcoding process is to reduce the total energy expended on the long bus.

We can envision many different transcoders. However, in keeping with a philosophy that the encoder and decoder are drop-in cells, we prefer techniques that do not change the timing of the bus. This simple goal rules out naive uses of compression that generate long, multi-cycle code words.

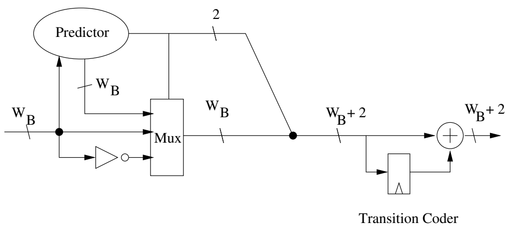



**Figure 2.** ***Utilizing Prediction for Transcoding:*** **The predictor generates a sorted list of up to $2^{W_B}$ possible values. These values are sorted by prediction confidence. On match, a code is generated to indicate which of the values matched. Two control bits select between predictions, raw data and raw inverted data.**

One simple enhancement to the scheme described above is to insert *transition coding* modules at either end of the bus. As shown by the figure, this enhancement causes the encoding of data from encoders and decoders to represent wire *changes* rather than absolute values: a one (1) on an output from the encoder represents a wire that will change its value (expend energy), while a zero (0) represents no energy expenditure. This change in representation greatly simplifies the energy optimization problem—even when accounting for cross-coupling between adjacent wires. For instance, we can easily perform *inversion coding* in which we send a value or its inverse to reduce transitions.

If some coding process reduces transitions on a bus, then we can conclude that it will *save energy* for *some bus length*. This is not a particularly sophisticated argument – energy consumption increases with increasing amount of communication and scales linearly with the length of the bus. For a given bus, type of traffic, and technology, we could say that there is a *break-even* length of the bus at which a transcoder can save energy. As technology sizes shrink, this bus length will shrink as well, since the power consumed by the encoder and decoder will decrease.

## 1.2. Using Data Prediction to Reduce Power

One interesting viewpoint for transcoder design is to utilize *value prediction techniques* [7, 19] – often viewed as providing limited performance enhancement – to save energy. In effect, we can run the same predictor on either end of the bus and compare its result with the actual value to be transmitted. Since both predictors are running synchronously and on identical values, they will provide identical predictions. When these predictions match with a value to be transmitted, we send nothing—to say that the prediction was correct. When they do not match, we send the original information directly (and transition a special control bit). Should the predictors attain a 100% prediction rate, then *no energy would be consumed sending information*. Many data prediction techniques, in particular value prediction, have reasonably high prediction rates [19], and if we can apply some of these techniques to the bus traffic, we could achieve a respectable reduction in energy by reducing the number of transition on the bus.

Figure 2 shows how we can use predictor confidence [8] to improve our technique. Assume that the set of possible predicted values are sorted by confidence. These values may come from a single prediction strategy, or a complex combination of multiple prediction strategies. The highest confidence value will be matched with a code word that has the lowest energy cost. Given transition coding, this would be the all zero vector (no transitions). The next $W_B$ values could be matched with the unique vectors of Hamming weight one (i.e. with a single bit set). After this, we would have to start using vectors with higher Hamming weight, *possibly sorted to account for cross-coupling.* Whatever the case, the predicted values are assigned code words from the space of $W_B$-bit words. When the predictor is presented with a new input word, the predictor checks its list of encoded values. If any of them match, it send the corresponding code. If none of them match, it will send either the data or its inverse.

## 1.3. Our Contributions

Previous studies of on-chip bus compression suffer from two deficiencies. First, they utilize random traffic, a poor approximation to real traffic. This has a tendency to underestimate the potential advantages of compression technology. In contrast, we examine traces of internal traffic from several buses in a simulation of a running superscalar processor. Our unique spin on this study will be to consider data prediction technologies as a foundation for transcoding.

Second, previous studies have focused on the reduction in bus traffic *while completely ignoring the complexity and energy consumed by the encoding and decoding circuits.* Two very important questions can only be answered by attempting to design a complete transcoder circuit:

1. Will the transcoder power be so high that no reasonably-sized chip will meet the break-even point?
2. Is the area consumed by transcoder logic too large for practical use?

We design a transcoder in a $0.13\mu m$ silicon technology, carefully producing a compact, energy efficient layout. We discuss some of its internal circuits, consider its size, and provide an accurate analysis of its potential for saving energy.

Our results show that despite potentially large reductions in bus energy achieved by more complicated prediction methods, a very simple, energy efficient transcoder is the only method which can effectively save energy for on-chip buses of reasonable length. With the transcoder of Section 5, we show that we can *break even* at length scales of less than 11.5mm at $0.13\mu m$ and a projected length of 2.7mm for a more complex design in $0.07\mu m$.

The rest of this paper is as follows: Section 2, presents related work. Section 3 presents a detailed model of long wires for modern chips. Then, Section 4 explores the potential savings that could be gained through prediction technologies, using SimpleScalar [6] and SPEC benchmarks. Section 5 follows with the architecture and layout of a practical bus compression engine. We describe future work and conclude in section 7.

## 2. Related Work

There have been a number of papers on the subject of coding for low power. Bus-Invert coding [23] and partial bus-invert coding [20] implement the same idea of inverting the bus value to be sent if more than half the wires are changing. [16] provides the circuits necessary to implement the scheme including a novel analog majority voter to count the number of ones on the input. Additional schemes include workzone encoding for address buses [15], which was extended in [1] to partition the memory space into a number of sectors that represent different segments of the address space. [13] proposes a similar design of sending the codeword xor'ed with the current input that has the lowest Hamming weight.

In addition to these experimental approaches, several papers propose more complicated algorithmic solutions. [21] proposes a static code-book but the encoding used is determined by the one that minimizes a more complex fitness function including inter-wire capacitance. [9] proposes a more complicated scheme for address buses that re-maps transitioning and non-transitioning wires to shield cross-coupled wires.

At the circuit level, Zhang et al. [25] proposes circuit enhancements to reduce the voltage swing on wire transitions in interconnect.

Basu et al. [4] proposes placing a value cache at both ends of a communication channel. When "hit", the system sends the index to the cache entry, instead of whole word, to reduce transitions. Their scheme focuses on off-chip buses and for DSP and embedded applications.

Parcerisa and González [18] applied value prediction technique for inter-cluster communication for clustered microarchitecture. However, their goal is to reduce long wire delay instead of energy.

Our technique differs from those listed above by incorporating value prediction techniques to reduce on-chip communication energy. In our paper, we also evaluate our energy reduction scheme against real bus traffic generated by SPEC benchmarks rather than randomly generated bus traffic.

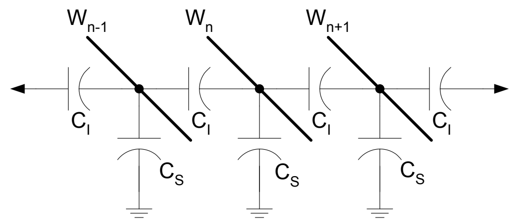



**Figure 3. *Wire Model:* Wires involve wire-to-substrate capacitance ($C_S$) and inter-wire capacitance ($C_I$). The ratio of the two ($\Lambda = \frac{C_I}{C_S}$) denotes the relative importance of *transitions* and *cross-coupling*.**

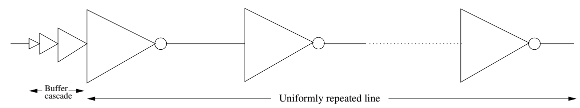



**Figure 4. *Repeater Model:* After an initial buffer cascade, repeaters are placed uniformly throughout wire.**

## 3. Interconnect Models

Our first order of business is to understand the characteristics of buses in modern integrated circuits. Energy consumption involves two distinct elements: *capacitances* between wires and the substrate, and *repeaters* to control latency.

### 3.1. Interwire Capacitance

Every wire transition expends energy. Furthermore, wires that are adjacent to one another expend energy through cross-coupling. A simple model that accounts for these two effects is shown in Figure 3 [21]. This figure illustrates two distinct types of capacitive couplings[1]: wire-substrate capacitance ($C_S$) and inter-wire capacitance ($C_I$). These values depend on technological effects such as the width and height of wires, oxide thickness, distance between wires, etc. Total capacitance grows *linearly* in the length of the data bus.

The relationship between expended energy and bus transition activity is governed by the equation for energy stored in a capacitor: $E = \frac{1}{2}CV^2$, where $C$ is the size of the capacitor and $V$ is the voltage stored on the capacitor. During the process of charging this capacitor, the power supply expends $2E$ total energy. We model the energy dissipation in two chunks: $E$ during the initial charge, and $E$ during the discharge. Consequently, the energy expended is *proportional* to the number of transitions (charge/discharge operations). In combination with the circuit of Figure 3, we can derive a model for the energy expended by wire $n$ ($W_n$), denoted by $E_n$:

$$E_n \propto L_{bus} \times (\alpha_n + \Lambda\beta_n) \quad (1)$$

1 We will ignore other parasitic effects (such as coupling between non-adjacent wires), since these are small in comparison [9].

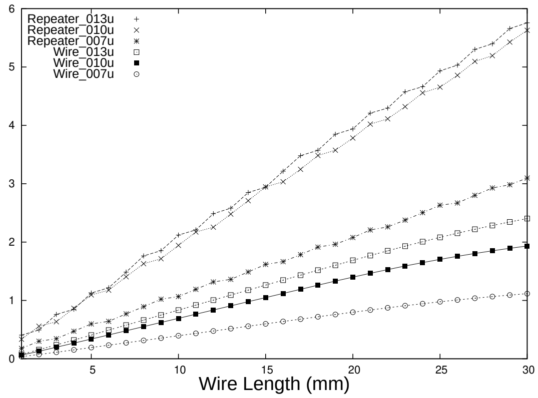



**Figure 5. Energy consumed by wire capacitance to substrate and adjacent wires for different technologies (L=1...30mm)**

Here, $L_{bus}$ is the length of the bus, $\Lambda$ is the ratio between inter-wire and wire-substrate capacitances (Figure 3), $\alpha_n$ is the total number of bus transitions on $W_n$, and $\beta_n$ is the total number of pair-wise inter-bus transitions between $W_n$ and $W_{n+1}$. Energy scales linearly with wire length because capacitance scales linearly with length.

$$\alpha_n = \sum_t (W_n^t \oplus W_n^{t+1}) \equiv \sum_t |W_n^t - W_n^{t+1}| \quad (2)$$

$$\beta_n = \sum_t |(W_n^t - W_{n+1}^t) - (W_n^{t+1} - W_{n+1}^{t+1})| \quad (3)$$

To explore the effectiveness of power reduction techniques, we compute values for $\alpha_n$ and $\beta_n$ over the course of some simulation. We can attack either or both terms as a way to reduce the energy of communication.

### 3.2. Signal Repeaters

For longer on-chip wires, repeaters are placed throughout to reduce delay. Therefore, we introduce a standard buffered wire model [2, 3, 17] that will later be included in the energy savings analysis of the various coding methods.

Our buffered model is illustrated in Figure 4. This standard model involves uniformly placed inverters of equal size throughout the length of the wire. An exponentially-increasing cascade drives the sending end of the wire. The ideal size and number of repeaters are technology-dependent. Consequently, we derived wire parameters as discussed in [12], including real technology parameters[2]. The initial drivers are needed because the size of repeaters are large (40 to 50 times wider than minimum size inverters). Although there are alternative approaches for repeater placement [22], this repeater scheme is the most commonly used at this time.

2 Wire parameters (resistance, capacitance and inductance) were derived from the Berkeley Predictive Technology Model (BPTM) [5], using wire geometries and dielectric values from the International Technology Roadmap for Semiconductors (ITRS) road map. The wires are placed at minimum pitch apart for this study.

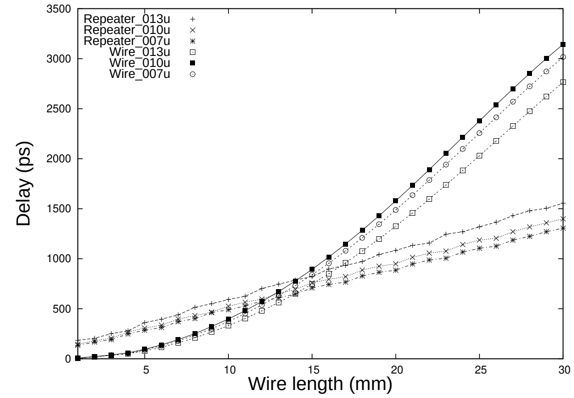



**Figure 6. Average wire propagation delay as a function of wire length (L=1...30mm) given our technology parameters**

| Technology | Wire type | Average $\Lambda$ |
|---|---|---|
| 0.13um | Unbuffered wire | 14.0 |
| | With repeaters | 0.670 |
| 0.10um | Unbuffered wire | 16.6 |
| | With repeaters | 0.576 |
| 0.07um | Unbuffered wire | 14.5 |
| | With repeaters | 0.591 |

**Table 1. Effective $\Lambda$ values for various technologies.**

The energy and delay of our wire model for various technology is shown in Figure 5 and Figure 6. These graphs are generated with HSPICE simulations using real process parameters (for $0.13\mu m$) and BPTM parameters (for $0.10\mu m$ and $0.07\mu m$). The number of repeaters to place varies with length of wire and is calculated based on [12]. Energy and delay for unbuffered wires are also included for comparison. Unbuffered wires exhibit quadratic delay with respect to length whereas the standard repeater model delay is linear. The buffered wire consumes more energy due to large repeaters.

Based on the energy consumptions from the simulations, we show the average effective $\Lambda$ for buffered and unbuffered wire in Table 1. Although inter-wire capacitance, $C_I$, is significant in long wires, its effect is less pronounced in buffered wires because repeater capacitances increases $C_S$.

## 4. Bus Traces and Compression

In this section, we describe the bus traffic we investigate for energy reduction. We briefly look at some of the statistical characteristics of this traffic. From here, we present some coding schemes and determine how well each of them reduce wire energy, taking into account coupling transitions.

### 4.1. Extracting Realistic Bus Traffic

To explore traffic along realistic buses, we instrumented SimpleScalar 3.0 [6] to capture data values on internal buses. SimpleScalar is a well-known *functional* simulator for out-of-order execution. In functional simulation, the results of instructions are computed immediately upon in-

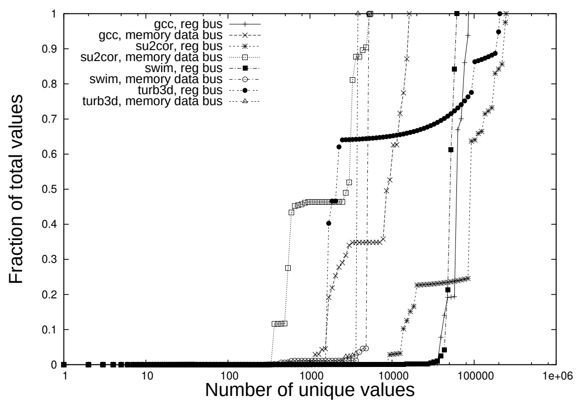



**Figure 7. CDF of most frequent unique values that appears in traces (for memory data and register buses)**

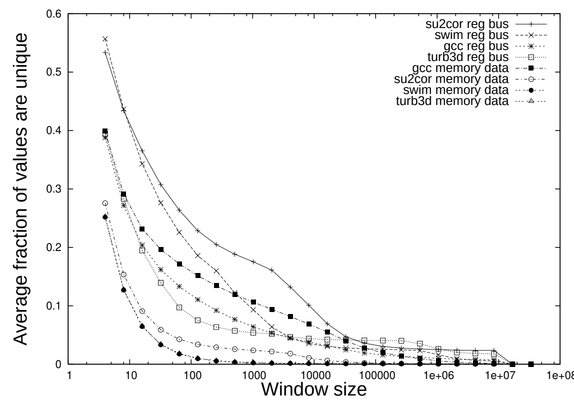



**Figure 8. Average fraction of unique values to total values found within some window size (for memory data and register buses)**

struction dispatch, with simple accounting to track input and output dependencies through registers and memory. User defined latency parameters are used to determine when individual dependencies can be resolved. As a result, there are no "buses" with realistic timing in SimpleScalar. To address this problem, we enhanced SimpleScalar with *bus timing generators* that extract values from the ongoing simulation and re-timed them to resemble actual bus timing.

In this paper, we explore two buses: the *memory bus* and the *integer register bus*. The memory bus tends to have high capacitance due to the fact that it extends off-chip or, in the case of Systems on a Chip (SOC), travels a long distance on chip. The integer register bus tends to have high fanouts, and thus may have high capacitance. These two buses are not the only long/capacitive buses in a microprocessor. Rather, they are intended to be representatives for evaluation of bus encoding.

*Output Bus to Caches/Memory:* The most detailed SimpleScalar simulator, sim-outorder, performs a fair amount of timing and dependency accounting. To simulate the external data and address buses, we maintained a queue of time-ordered entries for the value on the bus from the current cycle into the future. Memory events are inserted in the scheduler queue when load or store instructions are ready to execute. If the data must be fetched from main memory, the access latency calculated by sim-outorder generates an event corresponding to a future cycle in which the value will appear on the data bus. We refer to this bus as the "memory bus" throughout the rest of the paper.

*Register file output to functional units:* The synthesis of bus behavior was easy for the register bus, since every instruction goes through an explicit pipeline stage in SimpleScalar. Hence, we could easily determine what value would be on the register bus each cycle. We refer to this bus as the "register bus" throughout the rest of the paper.

## 4.2. Trace characteristics

To evaluate bus value compression effectiveness, we start by investigating some statistical properties of the bus values themselves. We pick some statistics which would shed light on how a few particular compressors might perform and use this to motivate a more detailed investigation later in this section of specific coding mechanisms.

Figure 7 shows the cumulative distribution function of all unique values, sorted in order of most frequent to least frequent. This graph attempts to show how many unique values make up the majority of trace values for several benchmarks from a 10 million value trace. For the 4 benchmarks we chose to look at in this analysis, none of them have a unique value set size with significant coverage until we get into the 100-1000 value range. This suggests that a strictly frequency-based compression approach will not be very effective unless we can afford a very large dictionary size to hold 100s-1000s of unique bus values.

Figure 8 shows the average fraction of values in a trace that are unique in a window, given a particular window size. This statistic shows that a Window-based dictionary compressor might have some success with even a small number of unique entries (10s of entries) because the fraction of values in a window are relatively small for even a 10 entry window.

## 4.3. Coding schemes

Now that we have seen a number of bus traffic characteristics, we delve into bus coding techniques. In this section, we look at a number of quasi-stateless and stateful coding mechanisms and then evaluate their effectiveness at reducing bus energy.

*Spatial encoder:* If $W_C = 2^{W_B}$, then we can arrange to take the input value and code it as a single value on the $2^{W_B}$-bit bus. This means that every possible input value causes at most one transition on the bus. We call this the "Spatial Encoder" since it converts each input value to a transition at

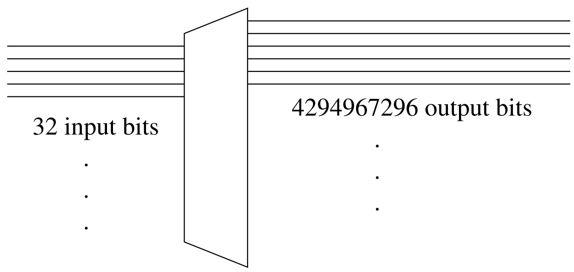



**Figure 9.** ***Spatial coder:*** **This stateless coder is represented as a demultiplexer**

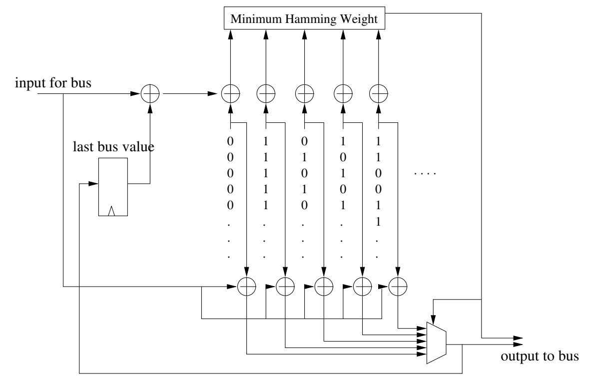



**Figure 10.** ***Generalized inversion coder:*** **This stateless coder chooses a bit vector that will minimize total and coupled transitions for a given input and current bus value**

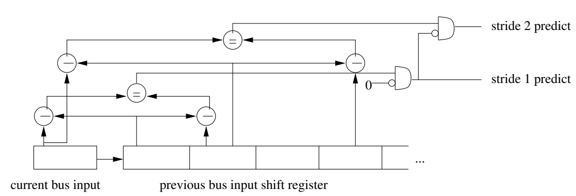



**Figure 11.** ***Stride predictor:*** **Block diagram of a simple stride predictor. Strides of one and two are calculated in the diagram but generalizes to any number of strides**

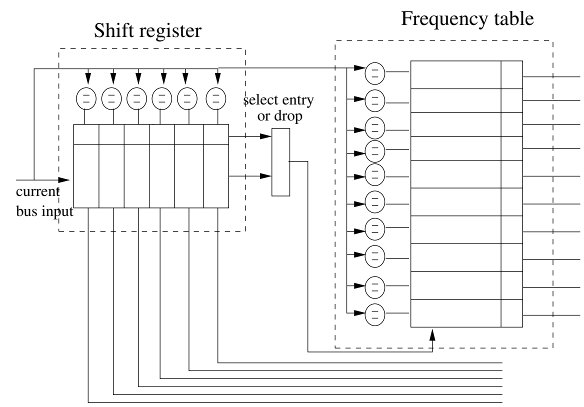



**Figure 12.** ***Context-based coder:*** **Input compared to entries in shift register and frequency table. On match, index of entry sent. Otherwise, new value shifted into shift register and exiting value dropped or replaces LFU entry in frequency table.**

a particular spatial location in the long bus. This provides extremely low communication energy at the expense of an impractical, exponential cost in area. In Figure 9, the spatial encoder is represented as a demultiplexer.

*Inversion encoder:* A more complicated stateless encoder is the generalized inversion coder in Figure 10. This general inversion encoder first produces a "transition vector" by xor'ing the current and new bus value, and then xors this transition vector with one of a number of available constant bit patterns to generate the next bus value. The bit pattern chosen is the one that results in the minimum total and coupled transitions after xor'ing. The computed bus value is sent along with a number of bits to identify which constant bit pattern was chosen allowing the decoder to reproduce the original data.

*LAST-value predictor:* A LAST-value predictor [14] captures strings of repeated values. Although we do not use this predictor by itself, we include it in combination with all of the remaining predictors. We assign code "0" to repeated values to avoid a penalty relative to the un-encoded case (which expends no energy when the bus is unchanged).

*Strided predictor:* Our first viable predictor contains multiple stride predictors and makes use of prediction confidence. As shown in Figure 11, a shift register containing previous bus values is used to calculate the stride of every data-word, every other data-word, every 3rd data-word, etc. The lower order strides are encoded with lower weight codes because they are assumed to be more frequently matched (having a constant stride every 4th cycle is usually more probable than every 67th cycle). Thus the predictors in which we have more confidence, are assigned a lower weight code. The lowest instruction interval to match the stride for the bus value to be encoded is used. The value sent is then just a indication of which strided predictor output should be used by the decoder.

*Window-based predictor:* Our second predictor captures the locality implied by Figure 8. It keeps the last few unique values in a shift-register and encodes them as low-weight

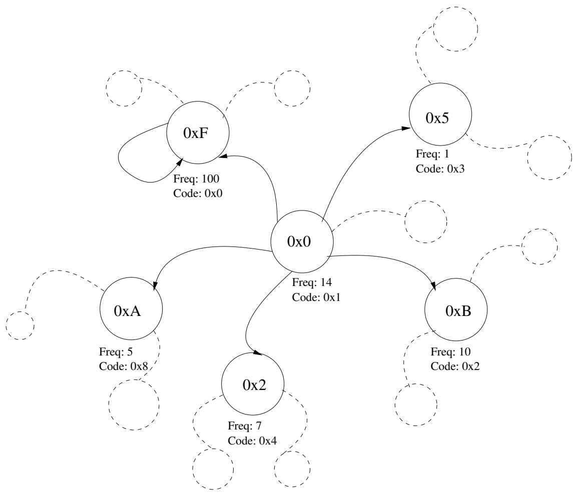



**Figure 13.** ***Value-based, Context-based predictor:*** **Coding based on the frequency of values. Here, states represent possible bus values, arcs represent transitions between them**

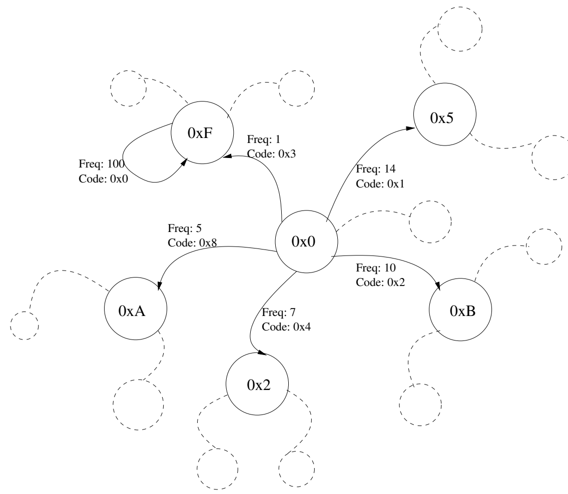



**Figure 14.** ***Transition-based, Context-based predictor:*** **Coding based on the frequency of transitions from value to value**

codes. A shift occurs when a value appears that is not already in the register, entering a new value and discarding the oldest value.

*Context-based predictor:* We considered two flavors of Context-based transcoder 1) value-based, and 2) transition-based. In the value-based transcoder, we perform transition activity compression by coding based on the frequencies of bus values. We maintain a table of frequently seen bus values and assign low Hamming weight codes to more frequent values [19] (see Figure 13). In the transition-based transcoder, we assign code based on the frequency of value transitions (*i.e.* number of identical consecutive inputs). We maintain a table of pairs of input values and assign low-weight codes to more frequent pairs (see Figure 14).

The schematics for the Context-based transcoder is shown in Figure 12. When the value to be sent is not in the table, the value is sent un-encoded. For table with number of entries less than bus width, the code-word associated with each entry can be a single bit since each wire in the bus can uniquely identify an entry. On the other hand, if the number of entries is greater than bus width, then some entries will have code-words with more Hamming weight.

Each entry has a frequency counter which is incremented on a match to the bus input. Naively, a new bus value could be inserted into the table immediately, replacing the lowest frequency value. However, this causes thrashing on lowest value. Instead, we utilize a Window-based predictor structure (shift-register) with frequency counts at the input. Entries in the shift register accumulate counts and are later entered into the frequency table if their frequency count is above threshold. To accommodate phases in computation, we introduced a periodic reduction in counter values; every

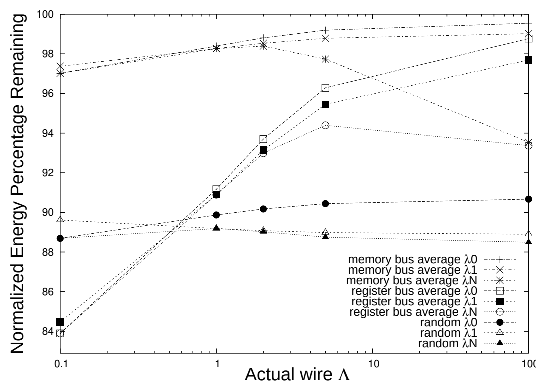



**Figure 15.** ***Inversion coder:*** **Normalized energy of coded wire as a function the wire's $\Lambda$ on the register bus for inversion coders which do not necessarily code according to the correct $\Lambda$**

so often, we divide all of the counters by 2. We call this period the "counter division time." Undivided frequency table entries could accumulate very high frequency counts long in the past which no longer represent relevant statistics of current bus traffic.

### 4.4. Coding effectiveness

With the coding schemes described, we are now going to determine the amount of bus energy each scheme can eliminate. Unless otherwise noted, we assume the transition to coupling energy ratio is 1 ($\Lambda = 1$). We also do not take into consideration the energy used in the encoding and decoding process. We will address this in Section 5.

*Inversion performance:* Figure 15 shows how well a number of these inversion encoders perform when using the following input streams: the register file output bus and the memory data bus from 4 SPEC benchmarks, and uniformly distributed random data. Three different minimizing func-

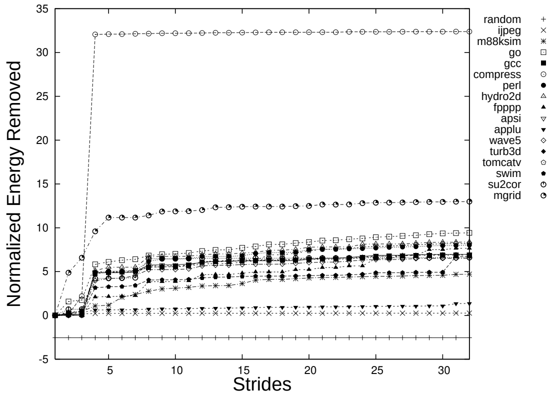



**Figure 16.** ***Strided Predictor Performance:*** **Percent of normalized energy removed by strided predictor on memory bus**

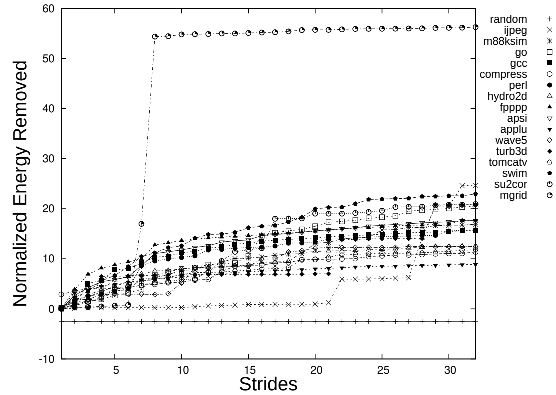



**Figure 17.** ***Strided Predictor Performance:*** **Percent of normalized energy removed by strided predictors on register bus**

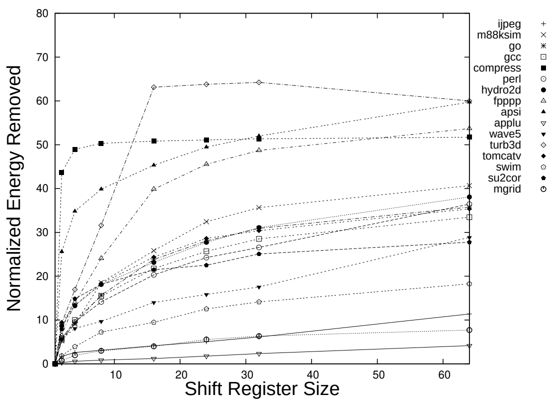



**Figure 18.** ***Window-based transcoder:*** **Normalized energy vs shift register size on memory bus.**

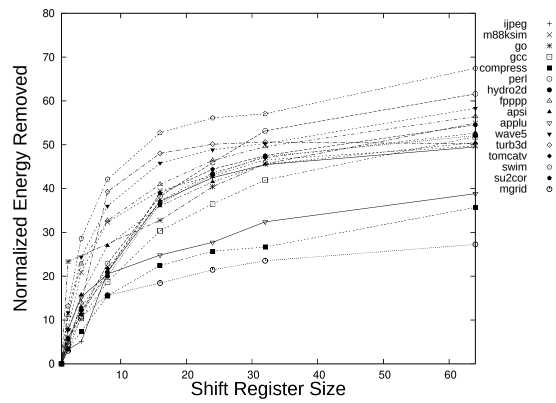



**Figure 19.** ***Window-based transcoder:*** **Normalized energy vs shift register size on register bus.**

tions are used: $\lambda 0$,$\lambda 1$, and $\lambda$N. In $\lambda 0$, the function chooses a bit pattern assuming the technology specific $\Lambda$ in equation 1 is 0, which is equivalent to the encoder in [23]. In $\lambda 1$, the function assumes $\Lambda = 1$, and in $\lambda$N, the function knows the correct value of $\Lambda$. We then ran the inversion coder with varying actual $\Lambda$ and calculate energy saved for each of the functions. Note that – except for high values of actual $\Lambda$, codes with measured $\Lambda = 1$ ($\lambda 1$) is pretty accurate approximation. The figure shows that with the exception of $\Lambda \leq .5$, using random data to determine the energy consumption of an encoding scheme will generally produce results that are better than what would occur in reality. This is why we evaluated our techniques on actual SPEC benchmark bus traces.

*Strided performance:* To evaluate the performance of the strided predictor, we look at how much energy expenditure (in the form of transitions and cross coupling events) is removed by having stride predictors available up to some interval. Figure 16 and 17 plot the energy reduced by the stride predictors, normalized to the un-encoded case, as a function of the number of stride predictors used. On the memory data bus in Figure 16, there is an observable jump in energy reduction between 3 stride predictors and 4 stride predictors but it is not large and the performance scales more or less linearly otherwise for most benchmarks.

With different bus traffic, like one of the register file output ports, Figure 17 again gives no obvious conclusion of how many stride predictors would be useful across the benchmarks. It is still the case though that adding more stride predictors reduces the number of bus transitions and thus the energy, which is what we would expect. These results show us that there probably is not an obvious number of stride predictors which would perform well over all workloads in many cases.

Thus, this stride predictor encoding scheme works reasonably well in reducing the energy expended by a bus. However, if we compare Figures 16 and 17 to Figure 15, we see that for the same bus and wire model, some of the stateless inversion coders remove more energy than the biggest stride predictors. This indicates that perhaps the stride predictors are not the best stateful coding mechanism.

*Window-based performance:* Figures 18 and 19 plot energy removed by Window-based transcoder, as a function of shift-register size. The knees of the curves center around 8. At this configuration, the transcoder removes about 19-

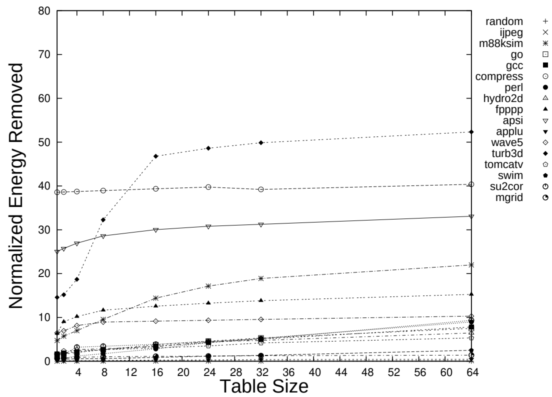



**Figure 20.** ***Context-based transcoder:*** **Normalized energy vs frequency table size on memory bus for transition-based design (shift reg size = 8)**

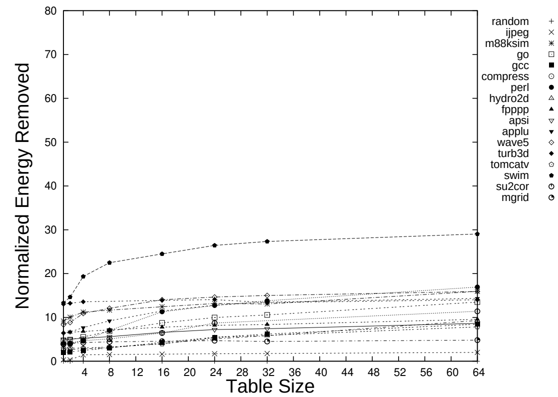



**Figure 21.** ***Context-based transcoder:*** **Normalized energy vs frequency table size on register port for transition-based design (shift reg size = 8)**

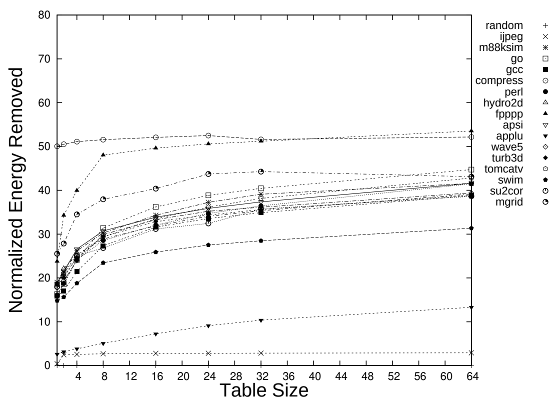



**Figure 22.** ***Context-based transcoder:*** **Normalized energy vs frequency table size on memory bus for value-based design (shift reg size = 8)**

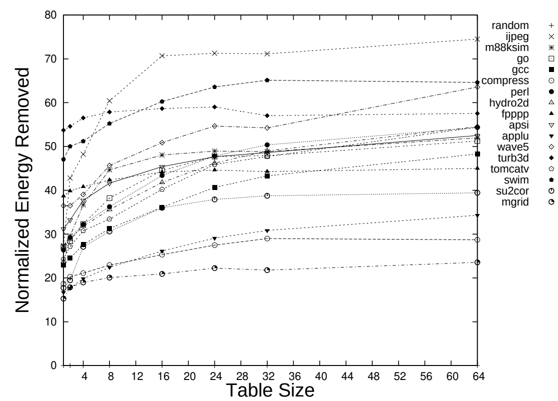



**Figure 23.** ***Context-based transcoder:*** **Normalized energy vs frequency table size on register port for value-based design (shift reg size = 8)**

25% of the energy—a respectable performance.

*Context-based performance:* Figure 20 and Figure 21 plot the energy removed by the transition-based transcoder, normalized to the un-encoded case. Comparing this with value-based transcoder (Figures 22 and 23), we see that the transition-based transcoder does not perform as well as value-based, given the same amount of hardware due to the fact there are many more arcs than states.[3] Because there are many more arcs, the probability is less that a given bus input will hit in either the frequency table or shift register compared against value-based scheme. Thus, we focus on value-based scheme for subsequent studies.

There are a number of parameters for the Context-based transcoder architecture. The primary ones being the length of the shift register, the counter division time, and the size of the frequency table. The length of shift register is important because it dictates how long a new value can accumulate frequency counts before it's shifted out. The counter division time determines how quickly the frequency counts are divided to accommodate phases of computation.

For frequency table sizes, Figures 22 and Figures 23 show that somewhere between 20 and 32 to be optimal for a shift register size of 8. These plots exhibit a more noticeable asymptote than the stride predictor simulations. We reach the point of diminishing returns for the frequency table size greater than 16. For shift register size, we found that 8 entries to be a good trade-off between normalized energy removed and hardware complexity, as shown in Figure 24. For division time, we experiment and find that the normalized energy removed levels off around 4096 cycles for many of the benchmarks and differing frequency table sizes, as shown in Figure 25.

From these result, the Context-based encoder removes

3 For 32-bit bus, there are $2^{32}$ states but almost $2^{64}$ arcs.

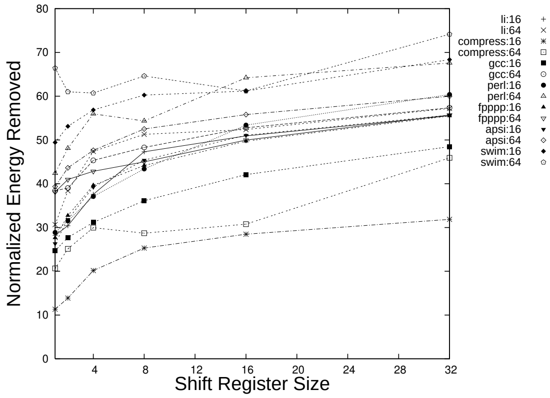



**Figure 24.** ***Context-based transcoder:*** **Normalized energy vs shift register size for the register output port with tables sizes of 16 & 64 for value-based design**

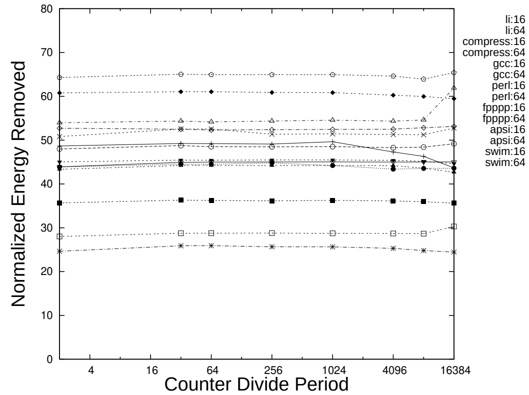



**Figure 25.** ***Context-based transcoder:*** **Normalized energy vs counter divide frequency for the register port with table sizes of 16 & 64 for value-based design**

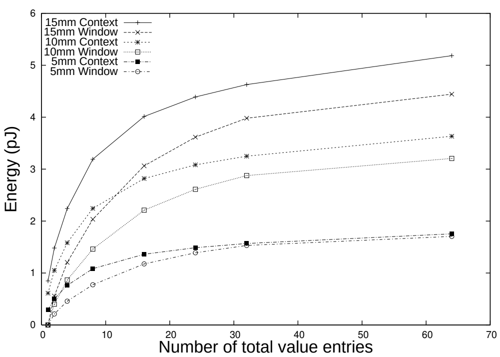



**Figure 26. Energy budget of a number of different wire lengths**

about 25% to 35% of normalized energy on average for reasonable shift register sizes (4 to 8) and table sizes (24 to 32). If we compare these results to the inversion coders and the stride predictors, the stride predictors remove only 10% to 15% of the energy and the inversion coders remove 15% to 20% (except for the random input).

The Window and Context-based transcoders have relatively little arithmetic logic compared to the other options. The inversion coder must compute up to 64 transition vectors and then decide the one with the smallest Hamming weight. This would involve 64, 32-bit xors for the transition vectors, accumulators for the weight calculation and comparators. Stride predictors require 2 subtractions and a comparison for each desired stride. The Context-based encoder requires only frequency comparisons and counters. The Window-based encoder is even simpler. Given their superior potential for removing energy, we continue to explore Window and Context-based encoders in following sections.

# 5. Toward Building a Real Transcoder

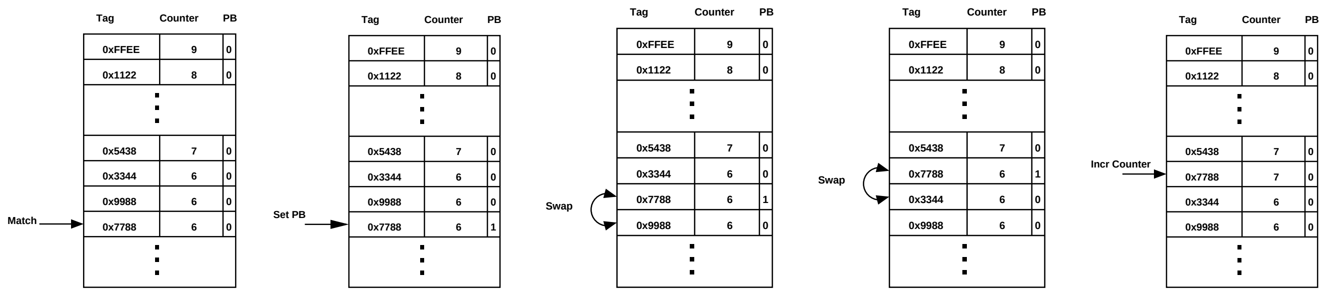
The very first task in designing a real transcoder is to determine how much energy we can use for encoding/decoding process. We called this metric, *energy budget*. Using such metric and careful analysis of energy consuming operations of the transcoder, we designed low-power circuitry to implement transcoder function and ultimately laying it out to determine final energy consumption. Because it is nearly impossible to simulate the entire circuit for the duration of SPEC benchmark runs in SPICE, we devised a statistical model to approximate the energy consumed by the transcoder. Furthermore, we estimated the energy consumption of transcoder for $0.10\mu m$ and $0.07\mu m$ technology using BPTM models. Finally we presented the crossover points[4] for various technologies and Window-based transcoder on two different buses.

## 5.1. Energy budget

If we consider the energy consumed by wire transitions of various coupling combinations and the transitions we eliminate using some coding module, we obtain what we will call our *energy budget*. This metric is independent of the particular implementation we choose to use for our encoding and decoding modules, and depends only on our particular wire model and transition code.

In Figure 26, we show the transcoder energy budget as a function of size in both Window-based and Context-based designs. The different lines correspond to particular wire lengths and transcoder configurations. Since transcoder energy is independent of wire length and each transition saved

4 Crossover points are defined as the wire length in which the energy of transcoder+wire equals to the energy of pure wire. It's an important metric because our transcoder will start saving energy from that point on.




(a) Entry “0x7788” is a match

(b) Pending bit for entry “0x7788” is set

(c) Entry “0x7788” and entry “0x9988” swapped

(d) Entry “0x7788” and entry “0x3344” swapped

(e) Counter for entry “0x7788” incremented, pending bit unset b/c “0x5438” has higher counter value than “0x7788”

**Figure 27.** ***Swapping Algorithm***

is worth more energy for longer lengths, we see that the energy budget increases with increasing wire length. Because the Context-based architecture saves only a small fraction more transitions than the Window-based architecture for shorter wire lengths, they have approximately the same energy budget. For longer wires ($\geq 15mm$), the energy budget gap between them widens.

In the following section, we are going introduce a simple base case, the inversion coder, for comparison. Then we are going to make a detailed implementation analysis of the transcoder designs to get an estimate on complexities, area and energy. We must choose one of the two transcoder designs for the energy budget analysis. Since both designs save about the same amount of energy, we consider a complexity metric to break the tie.

## 5.2. Base Case: Simple Inversion Coder

We examined a simple inversion coder to compare against the Context-based and Window-based transcoder. The inversion coder we studied is slightly different than those described in previous literatures [16, 20, 23]. Since our goal is to minimize transitions, instead of minimizing the Hamming weight based on input alone, we minimize the Hamming weight of the XOR of input and the current bus value. The reasoning for this approach is to minimize transitions for a string of repeated values (in which case, transition coding will perform worse than un-encoded case).

## 5.3. Transcoder Designs

In this section, we describe an efficient sorting algorithm, transcoder operations, and custom circuits for the Context-based and Window-based transcoder. We also discuss the complexities of these two designs.

**5.3.1. Efficient Sorting Algorithm** To avoid storing the codewords directly in the frequency table, we used the position of an entry in the table as the code. Thus it is important to ensure that the most frequent values in the frequency table are coded with minimum weight codes. Therefore, sorting the frequency table is necessary to keep the counter values in order. We devised a low-overhead sorting algorithm to keep frequency table sorted in Context-based transcoder. This algorithm avoids a full resorting upon every table update. Performing general sorting in hardware is non-trivial and requires a large amount of area. Most sorting scheme requires order $O(N \log N)$ comparators for the sorting network and/or a memory element [11, 24]. To avoid such overhead, we introduce the following invariants:

**Invariant 1:** Each entry must have an unique tag. Thus there can be at most one match for a bus value.

**Invariant 2:** Entries placed higher in the table have frequency counter values greater or equal to the entries located lower in the table.

Invariant 1 is used so we can use the match lines as the code word transmitted and we do not need to store the code words in the frequency table. This simplifies the actual encoding logic.

Invariant 2 is used to avoid the complexity of general sorting. To perform a full sort, each entry in the frequency table must be able to swap with any other entry arbitrarily and entry pairs must be able to perform full comparisons between the counter values[5]. The amount of wire connections goes up by $O(n^2)$. The wires will have large load capacitance and every swapping will cause large significant power consumption. Instead of allowing arbitrary swapping, we limit each entry to swapping with its neighbors only. Thus there are only $O(n)$ connections. This not only reduces load capacitances on these local wires, it also keeps their lengths short to further reduce wire capacitance. Implementing a full comparator requires more hardware and increases the complexity of the encoder. Thus we decide to use XOR comparators that only distinguish equal or not equal cases.

5 By full comparison, we mean that the comparison logic can distinguish that a value is greater than, equal to or less than another value.

The next challenge is to preserve sorted order under this restriction. We introduce a pending bit for each entry to accomplish this. The algorithm works as follows:

1 When there is a "hit" in the frequency table, instead of incrementing the counter of the "hit" entry immediately, we set the pending bit, indicating that the counter has a pending increment.

2 Every cycle, the top entry of the frequency table will have its counter incremented if its pending bit(s) are set.

3 Every cycle, pair-wise comparison between entries are performed. If the counter values are different between adjacent entries (*i.e.* the lower entry has a lower counter value than the higher entry), then we do not swap them. We increment the counter(s) if the respective pending bit(s) is set, and unset the pending bit. However, if the entry counters are the same and the pending bit is set for the lower entry, then we swap the two entries.

The reason we use the pending bit to indicate imminent increments is to handle the situation where there are two or more consecutive entries with the same counter values. In that case, if we increment the counters immediately and swap the two entries, we will break the invariant that entries lower have smaller counter values than higher entries. Figure 27 (a) to (e) illustrates how this algorithm preserves the invariant. In (a), entry "0x7788" was "hit". Thus, the pending bit is set in (b). Then the counter values of "0x7788" and "0x9988" are compared and found to be equal. In (c), the FSM swaps "0x7788" and "0x9988" because the pending bit of "0x9988" is not set. In (d), the FSM swaps, again, "0x7788" and "0x3344" since the pending bit of "0x3344" is not set. In (e), the, counter for "0x7788" is incremented and pending bit is unset since "0x5438" has a greater counter value than "0x7788". From these figures, it is clear that if the counter is incremented as soon as a "hit" occurs, invariant 2 cannot be preserved[6].

**5.3.2. Required Operations** To perform the coding and sorting functionalities, the Window-based and Context-based transcoder design needs a number of elementary operations which make up its dynamic power consumption. The operations are labeled in Figure 28 and are the following:

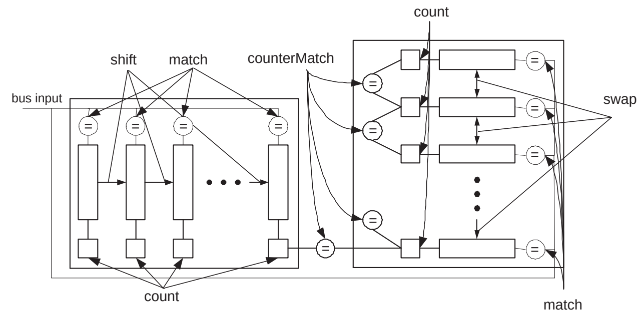



**Figure 28. Energy consuming operations in transcoder.**

**count:** Counters are necessary to track the relative frequency of a particular bus value. All lookup table and shift register entries must have individual counters which count up when the associated value is observed on the bus. Our design uses Johnson counters as an energy efficient counting method[7]. For each clock cycle, a single Johnson counter will increase.

**match:** The bus input value is compared to all the entries in the shift register and lookup table to determine whether we can send a dictionary index instead of the full value. To minimize the energy cost of these matchings, we make use of a selective precharge matching circuit [26].

**counter comparison:** We must compare counters to sort frequency table entries. An arbitrary sort is costly; instead, we *start* with all entries sorted by frequency. We maintain this invariant by catching situations in which adjacent counter values match and the lower value is incremented; we then swap these entries. Values from the shift register enter the table when they are more frequent than the least-frequent table entry.

**swap:** When two frequency table entries accumulate frequency counts such that the now less frequent value is still in the more frequent position, we must swap them. Swapping is accomplished through a combination of transmission gates and multiple clocks.

**shift:** A new bus value can be inserted into the shift register every cycle. The value on the end of the shift register shifts off the end and is either inserted into the frequency table or discarded based on its frequency count. The shift register entries do not actually change positions, rather the oldest value is replaced with the newest and a tail pointer is updated for each shift.

**last value tracking:** We must catch repeated strings of values to achieve LAST-value prediction (coded as "0").

**5.3.3. Customized Circuits** To implement the operations described above, we made use of some novel circuit tech-

6 One caveat of this algorithm is that "hit" to an entry with pending bit already set will be lost. However, this does not affect the correctness of the algorithm, only that some counter values might not be as high as it should be.

7 Johnson counters are energy efficient because for each count, only one bit will make transition. Furthermore, control logic for Johnson counters are trivial.

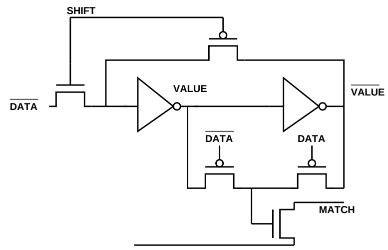



**Figure 29.** ***Low power CAM cell with shift:*** **This circuit minimizes the number of clock lines required for operation.**

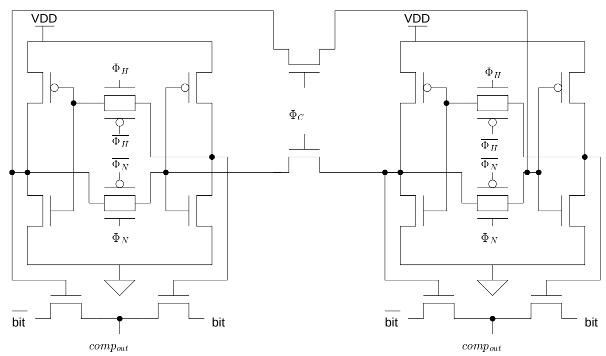



**Figure 31.** ***CAM Cells:*** **Circuit diagram of two neighboring CAM cells for Context-based transcoder. $\phi_H$, $\overline{\phi_H}$, $\phi_N$ and $\overline{\phi_N}$ control the connection between the cross-coupled inverters while $\phi_C$ controls the connection between two neighboring cells.**

niques to ensure lower power consumption for the encoder and decoder circuits.

**Johnson counters:** Instead of using binary counter, we employed a Johnson counter which is more energy and area efficient. For our hardware, we concatenated four 4-bit Johnson counters together to achieve a maximum count of 4096 ($8^4$) before saturation.

**Selective precharge matching:** If each new bus value had to probe all the values in the frequency table and shift register every cycle, this would result in many unnecessary charging and discharging cycles. Instead, we used a circuit which selectively precharged the lower order bits in the CAM cells [26]. Only if the lower order bits match do we charge the comparators of the remaining bits in an entry to complete the match. This reduce the number of unnecessary charging and discharging of all 32 bits table or shift register entry.

**Single clock shift cell:** As shown in Figure 29, we uses a single PMOS pass transistor to enable/disable feedback loop between the cross-coupled inverter. The advantage of this design over transmission gate is that no

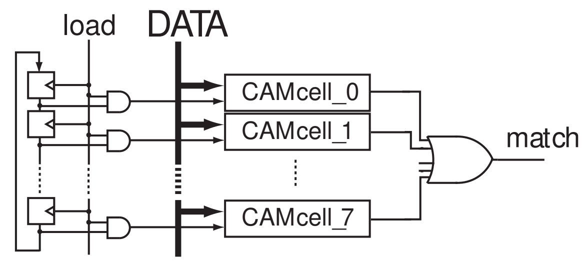



**Figure 30.** ***Pointer-based 8 entry shift register.*** **Instead of shifting tag value between entries, we kept a bit indicating the tail of the shift register. This reduces many bit changes within the tags**

complementary shift signal is needed and there is one less clock to route.

**Pointer-based shift entries:** We used pointer-based shift register design, as shown in Figure 30. Thus for every shift, only the value at head entry gets changed. This design is nice because it saves the number of bit transitions (therefore energy) on the shift registers for a shift operation.

**Pointer-based last value:** Since the last input value is in the shift register, we maintain a vector of bits (one bit per entry) to point at this value. This approach reuses the matching circuits for LAST-value functionality.

**Efficient entry swapping:** To reduce the energy used by the swapping of table entries for the sorting mechanism, the frequency table is composed of customized CAM cells that allow for efficient swapping between neighbors (see Figure 31). This circuit design is area efficient; requiring only two additional transistors for adjacent swapping. When two neighboring cells need to be swapped, $\phi_H$, $\overline{\phi_H}$, $\phi_N$, and $\overline{\phi_N}$ are disabled to break the positive feedback. After some delay, $\phi_C$ is enabled to connect the neighboring cells. At this point, the output of one inverter from each cell will write its value to the other inverter in the neighboring cell. Then $\phi_H$, $\overline{\phi_H}$, $\phi_N$ and $\overline{\phi_N}$ are re-asserted to enable the feedback path.

**5.3.4. Discussion** The frequency table adds considerable complexity to the Context-based design. If we compare the two designs with the total number of entries constant, we see that the Window-based design only requires shifting bus value entries and value matching circuitry as shown in Figure 28. When we add the frequency tracking of the frequency table, all value entries must be augmented with a Johnson counter, counter matching circuitry and the necessary control logic to determine swapping, counting and sorting. In a preliminary transistor level schematic of the full Context-based design, we noted that the counter and counter match circuitry takes approximately 33% of the circuit area.

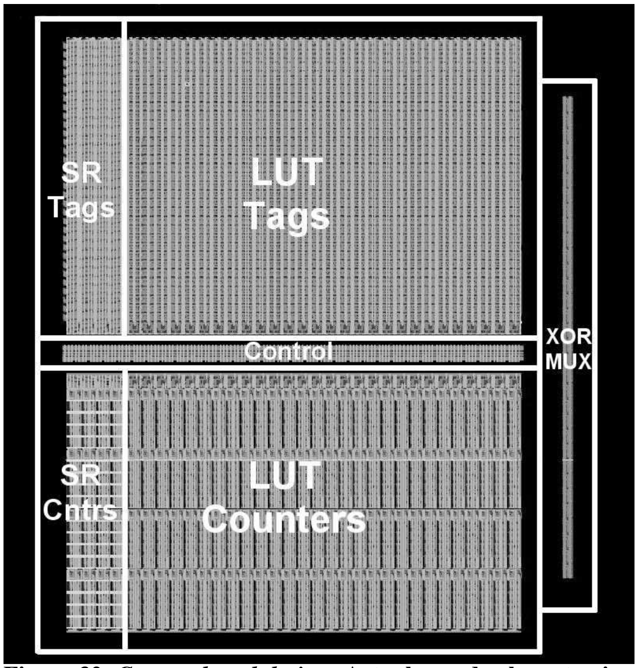



**Figure 32.** ***Context-based design:*** **Actual encoder layout using 0.18$\mu m$ process. The width and height for this layout is 400$\mu m$ by 500$\mu m$**

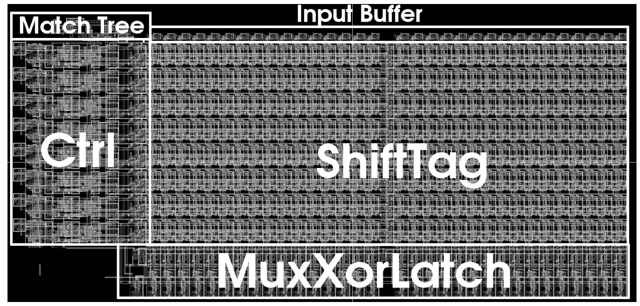



**Figure 33.** ***8 entry Window-based design:*** **Encoder layout using 0.13$\mu m$ technology. The width and height for this layout is 165$\mu m$ and 75$\mu m$**

In the following section, we are going to describe and evaluate the physical implementations of the transcoders.

## 5.4. Evaluating the Implementation

We implemented the Inversion coder, Context-based and Window-based transcoder in layout to evaluate accurate energy consumptions.

### 5.4.1. Three Designs

*Inversion Coder:* We constructed a version of inversion coder in layout using standard cells to compare against the Context-based transcoder. A major component of any inversion coder is a majority voter that will count the number of “ones” in its input and assert “1” if it’s majority. [16] describes a novel analog majority voter circuitry using current mirrors for 8 bits. However, after extensive simulation, we found such design unsuited for wider buses (*i.e.* 32 bits) due to low noise margin and constant direct current. Thus we reimplemented the majority voter using carry-save adder design [10] to add up the number of “ones”. The energy expenditure of this design will be described later in this section.

*Context-based:* The Context-based design implements “Johnson counters”, “selective precharge matching”, “efficient entry swapping” circuit techniques and the sorting algorithm. We laid out the encoder using 0.18$\mu m$ process as shown in Figure 32. The encoder is 400$\mu m$ high by 500$\mu m$ wide, occupying an area of 200000 $\mu m^2$.[8] The decoder should take up almost equal area since it shares similiar designs as encoder. Most of the area is taken up by tags and counters; the actual control takes very little area. Data comes in to the input buffers at left and either the code or the original data emerges from the right. This layout contains 4 shift register entries and 28 frequency table entries, the maximum number of entries for 1 bit code. The control is distributed across the width of the layout between the tags and counters.

*Window-based:* The Window-based design implements “single clock shift cell”, “pointer-based shift entry” and “pointer-based last value tracking”. It does not need to implement the sorting algorithm because there are no counters to sort. The physical layout was done using ST Micro’s 0.13$\mu m$ process as shown in Figure 33. The encoder is 75$\mu m$ high by 165$\mu m$ wide, taking up an area of 12400$\mu m^2$. In this layout, data goes into the input buffer at top and output at bottom. The ShiftTag section stores the unique 8 values and with embedded matching logic. The control logic to the left decides whether to send codeword or actual data on the bus and also determines the entry to shift out when a new value comes in. The MuxXorLatch at the bottom implements transition coding.

8 Even with a first-order scaling, it will still take up almost 100000 $\mu m^2$ in 0.13$\mu m$.

| Technology | Voltage (V) | Area ($\mu$m$^2$) | Op energy (pJ) | Leakage (pJ) | Delay (ns) | Cycle Time (ns) |
|---|---|---|---|---|---|---|
| $0.13\mu m$ | 1.2 | 12400 | 1.39 | 0.00088 | 3.1 | 4 |
| $0.10\mu m$ | 1.1 | 7340 | 1.07 | 0.00338 | 2.4 | 3.2 |
| $0.07\mu m$ | 0.9 | 3600 | 0.55 | 0.00787 | 2 | 2.7 |
| InvertCoder | 1.2 | 4700 | 1.76 | 0.00055 | 2.2 | 2.2 |

**Table 2. Leakage energy, delay and cycle time of transcoder for various technologies. Note: Delay is measured as data ready to bus out; leakage energy is per cycle. The area for 0.10$\mu m$ and 0.07$\mu m$ technologies are scaled based on 0.13$\mu m$ area. The voltage is based on ITRS roadmap projected values. The InvertCoder is constructed using 0.13$\mu m$ technology.**

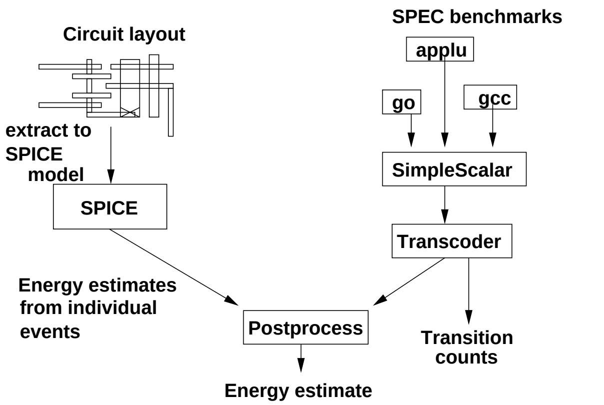



**Figure 34.** ***Experimentation methodology:*** **SimpleScalar communicates selected bus values to transcoder. Transcoder outputs the raw data, including transition counts and actual hardware operations performed. The output is then post-processed with input from SPICE simulation of the actual layout to produce energy estimates**

**5.4.2. Methodology** To evaluate the layouts, we first extract them to SPICE netlists and simulate for cost of energy operations under HSPICE. The energy costs of these operations are then combined with actual number of operations performed for a particular benchmark to obtain full energy expenditures.

The simulation flow used in this paper is shown in Figure 34. We instrumented SimpleScalar to output various bus values every cycle and feed the value into our transcoder simulator. The output of the transcoder simulator is then post-processed to obtain transition and energy expenditures.

The ultimate goal of transcoding is to reduce the energy consumed by buses. Thus it is critical to determine how much energy is used to perform the encoding and decoding process for an actual layout. The ideal and most accurate method for determining the power consumption of the coding circuitry would be to make a physical layout of the circuit and simulate it for power usage on the buses for the previously mentioned benchmarks. This would have been too slow.

Instead, as shown by Figure 34, we gathered statistical averages of various operations in the high-level transcoder module. This module closely simulates the hardware transcoder architecture and keeps a running total of the energy consuming operations performed by this architecture. These numbers are later combined with energy dissipation numbers derived from HSPICE simulations of an actual extracted layout netlist. Based on these energy numbers and operation counts, we derive total energy expenditure.

We validate the derived total energy expenditure against energy obtained by running the layout netlist with a short 100 cycle trace. The derived energy comes within 6% of the actual energy expenditure. This method, although less accurate than simulation of the complete layout, achieved tremendous increase in simulation speed.

To obtain estimates of energy usage from our transcoder circuit layout with future technologies, we used a number of tricks to scale the layout extraction SPICE netlist. We used the Berkeley Predictive Technology models [5] for $0.10\mu m$, and $0.07\mu m$ processes. We used the less accurate BPTM technology models because ST Micro models were not available for other process technologies.

We performed the technology scaling through the following procedure: (1) transistor gate lengths and widths, along with source/drain peripheral lengths are scaled linearly by the quotient between desired and current minimum feature sizes. Source and drain areas are scaled quadratically. (2) Wire capacitances are scaled based on the Berkeley Predictive Technology Model estimates for interconnect dimensions and their parasitic capacitor model. (3) This modified extraction netlist was simulated using HSPICE to find energy consumption of the scaled circuit.

**5.4.3. Results** Given the discussions from previous sections, the Context-based design requires at least 33% (more in layout) more area, more global clocks and more complicated control than Window-based design. Since the gain in terms of energy budget for the Context-based design is small for relatively short wire lengths, we decided to investigate the Window-based transcoder in depth.

*Inversion coder:* Using the simulation methodology described in previous section, we found that, on average, the inversion coder consumes 1.76 pJ per cycle. This is inadequate to break-even, even at 30mm because the inversion coder removes less transitions than our transcoder. The characteristics of the inversion coder are presented in Table 2.

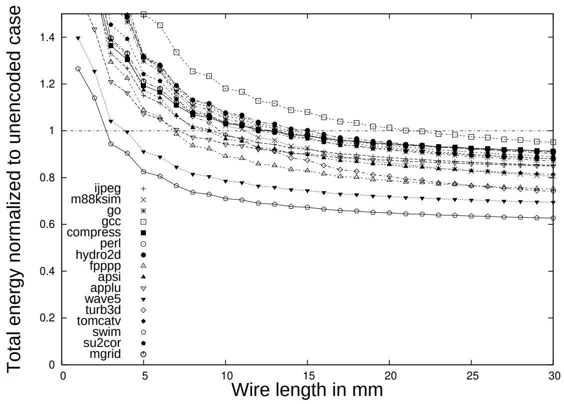



**Figure 35. *Window-based Performance:* Energy expenditure vs bus length on register bus (Shift register size = 8)**

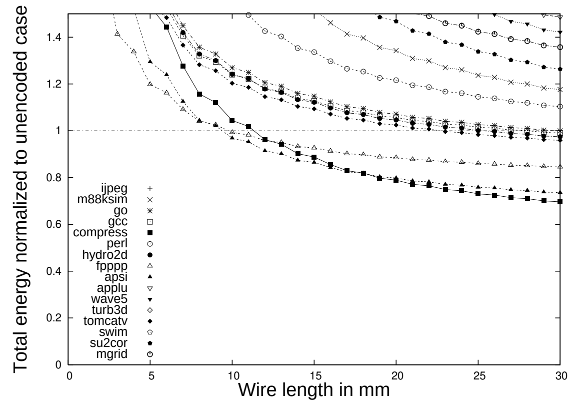



**Figure 36. *Window-based Performance:* Energy expenditure vs bus length on memory bus (Shift register size = 8)**

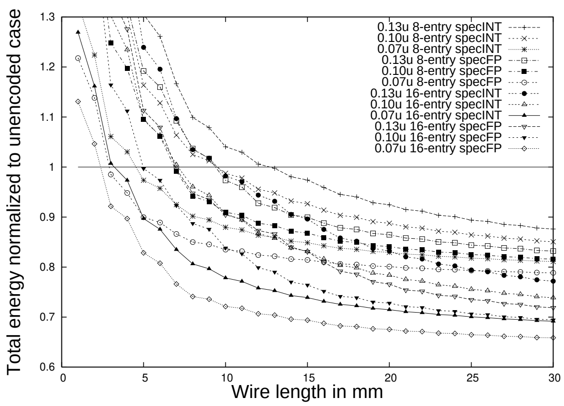



**Figure 37. *Crossover Length:* Wire lengths at which energy used by Window-based transcoder equals energy saved by coding on register bus for current and projected technologies.**

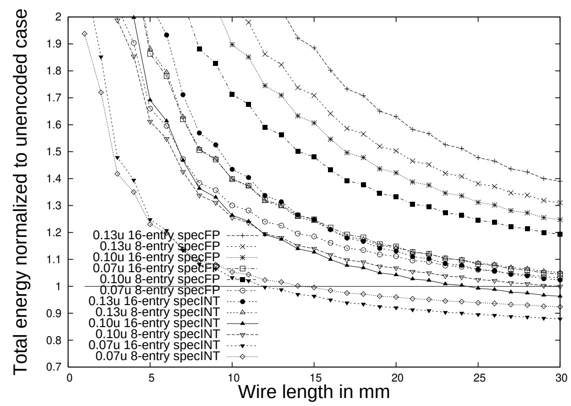



**Figure 38. *Crossover Length:* Wire lengths at which energy used by Window-based transcoder equals energy saved by coding on memory bus for current and projected technologies.**

*Window-based:* We present the impact our transcoder have on real circuits in Table 2. Even though leakage energy increases as technology shrinks, it is still orders of magnitude smaller than dynamic energy of the encoder. It is because our encoder small and consists of less than 5k transistors. The delay for the encoder is high due to the serial NAND match design. This delay could be reduced by making optimizations for speed in the matching circuit, and adding a small amount of additional power. The matching circuit is currently made of two NAND trees of 16 bits each, but one could imagine breaking this tree into more groups of smaller numbers of bits or changing it into a flatter OR-based matching circuit. Additionally, due to limited manpower, we were unable to fully optimize transistor sizings in the swapping circuits. A better designed version would have less latency.

To obtain energy expenditure, we utilized parameters obtained from the spice simulation of an actual layout for Window-based design. We also utilized various scaling methods to obtain statistical energy expenditure estimates for a number of different device technologies.

We obtained the following energy graphs for SPEC95 benchmarks on memory and register bus for various wire lengths with Window-based design. Figure 35 and 36 shows the ratio of transcoder+wire energy versus pure wire energy with 8 entry shift register design. The 8 entry design performs fairly well on register bus, with energy savings on almost all benchmarks at wire length greater than 15mm and median at around 11.5mm. However, for SWIM, the transcoder begins to save energy as short as 3mm. The result is less encouraging for memory bus. This is due to the fact that the absolute number of transitions removed is low (even though fraction of transitions reduced were high). Thus energy removed on wire transitions was not enough to offset the transcoder circuitry energy. Perhaps a different coding scheme with simpler encoder is needed to save wire transition energy on memory bus.

| Technology | Entries | SPECint | SPECfp | ALL |
|---|---|---|---|---|
| $0.13\mu m$ | 8 | 12.7mm | 9.4mm | 11.5mm |
| | 16 | 9.5mm | 6.9mm | 7.0mm |
| $0.10\mu m$ | 8 | 9.5mm | 6.9mm | 8.0mm |
| | 16 | 7.1mm | 5.0mm | 6.4mm |
| $0.07\mu m$ | 8 | 4.5mm | 2.9mm | 4.1mm |
| | 16 | 3.2mm | 2.4mm | 2.7mm |

**Table 3. Median crossover lengths for Window-based design.**

Figure 37 and 38 shows the scaling trend for $0.10\mu m$ and

$0.07\mu m$ technology. It also includes trends for a projected 16-entry transcoder design. The lines represents the median of SPECint and SPECfp benchmarks with different technologies and transcoder designs. The resulting crossover lengths are given in Table 3. As technology shrinks, the crossover point becomes shorter, which is what we would expect since wire power grows more dominant in smaller technologies. The crossover points for projected 16-entry transcoder are also shorter because a 16-entry transcoder removes more energy from wire transitions. This trend signifies that as technology shrinks, larger transcoders are well-positioned to take advantage of the growing disparity between wire and device energy.

## 6. Future Work

The fixed length code as presented here is simple to implement and simple to transmit since it does not modify data transmission timing. However, it is not the most efficient coding scheme by far. Variable coding can achieve greater compressibility in space and time. This in turn can actually reduce the overall energy consumption over a window of time[9]. However, the hardware complexities to produce variable length codes are considerably greater than our coding scheme. Furthermore, such variable-length coding scheme will change transmission timing on the bus, further complicating designer's task to incorporate encoder/decoder. That said, it is still interesting to investigate how well variable-length coding scheme works and compare it to our transcoding scheme.

We make the argument here that as technology continues to shrink, the use of the transcoder will become more attractive due to smaller encoding and decoding circuits relative to the bus length. Furthermore, our extrapolated energy consumption based on BPTM SPICE models does show better energy-saving with smaller technologies. However we would still like to perform more exact physical level simulations of the entire bus/encoder/decoder system with different process technologies to observe how the total energy usage scales.

## 7. Conclusion

We have briefly explored the space of prediction and compression of wire transitions, evaluating the relative performance for a number of examples. It has been shown that these techniques are indeed effective in reducing energy consuming wire events. We achieved an average of 36% transition reduction for SPEC95 benchmarks on register bus. We by no means performed a complete exploration of the bus value compression design space but we endeavored to give a detailed look at a number of disparate possibilities. From this high level evaluation, we pushed all the way down to a silicon implementation of the context-based transcoder. Using this physical model, along with a number of simulation tools, we performed a holistic evaluation of the transcoder's power consumption and bus power savings for many SPEC benchmarks.

We found that the full encoder/decoder, particularly the 8 entry Window-based transcoder, does indeed save energy for almost all SPEC95 benchmarks at wire length greater than 15mm and median at around 11.5mm at $0.13\mu m$ for register bus. Projection of a 16-entry design at $0.07\mu m$ breaks-even at wire-lengths of only $2.7\mu m$. Through our comprehensive energy usage evaluation, we believe that trading logic complexity to save on-chip communication energy will be increasingly attractive as Moore's law marches forward.

9 Even though instantaneous power might be greater, the transmission is done in shorter period of time. Furthermore, there are actually less bits to be transmitted.

## 8. Acknowledgements

I owe a great deal of thanks to my fellow colleagues who helped me with simulations, brain-stormed ideas and provided wonderful suggestions: Mark Whitney, Yatish Patel and Nemanja Isailovic. I also like to thank members of the IRAM group – Joseph Gebis, Christoforos Kozyrakis and Sam Williams – for their generous support on my initial physical layouts. I also like to thank Berkeley Wireless Research Center (BWRC) for allowing me access to technologies for which the final implementation is based upon. Finally, I thank my research advisor, John Kubiatowicz, for his insightful guidance and support, without which this work would not have been possible.

## References

[1] Y. Aghaghiri, M. Pedram, and F. Fallah. Reducing transitions on memory buses using sector-based encoding technique. In *Proceedings of the 2002 international symposium on Low power electronics and design*, pages 190–195. ACM Press, 2002.

[2] H. B. Bakoglu. *Circuits, Interconnections, and Packaging for VLSI*. Addison-Wesley, 1990.

[3] H. B. Bakoglu and J. D. Meindl. Optimal Interconnection Circuits for VLSI. *IEEE Transactions on Electron Devices*, pages 903–909, 1985.

[4] P. J. Basu K., Choudhary A. and K. M. Power protocol: reducing power dissipation on off-chip data buses. In *Proceedings 35th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-35). IEEE Comput. Soc. 2002, pp.345-55. Los Alamitos, CA, USA.*, November 2003.

[5] Berkeley Predictive Technology Models. http://www-device.eecs.berkeley.edu/˜ptm/.

[6] D. Burger and T. Austin. The Simplescalar Tool Set Version 2.0, 1997.

[7] B. Calder, G. Reinman, and D. Tullsen. Selective Value Prediction. In *ISCA*, 1999.

[8] D. Grunwald and A. Klauser. Confidence Estimation for Speculation Control. In *ISCA*, 1998.

[9] J. Henkel and H. Lekatsas. $A^2BC$: Adaptive Address Bus Coding for Low Power Deep Sub-Micron Designs. In *ACM Design Automation Conf.*, 2001.

[10] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach, second edition*. Morgan Kaufmann, San Francisco, California, USA, 1995.

[11] C. Huang, G. Yu, and B. Liu. A Hardware Design Approach for Merge-sorting Network. In *IEEE Int'l Symp. on Circuits and Systems*, 2001.

[12] Y. Ismail and E. Friedman. Effects of Inductance on the Propagation Delay and Repeater Insertion in VLSI Circuits. *IEEE Transactions on VLSI Systems*, 8(2):195–206, 2000.

[13] S. Komatsu, M. Ikeda, and K. Asada. Adaptive Codebook Encoding for Low-Power Chip Interface. *Electronics and Communications in Japan II*, 83(1):17–23, 2000.

[14] M. Lipasti, C. Wilkerson, and J. Shen. Value Locality and Load Value Prediction. In *ASPLOS*, 1996.

[15] E. Musoll, T. Lang, and L. Cortadella. Exploiting the locality of memory references to reduce the address bus energy. In *Proceedings of the 1997 international symposium on Low power electronics and design*, pages 202–207. ACM Press, 1997.

[16] K. Nakamura and M. A. Horowitz. A 50% noise reduction interface using low-weight coding. In *Symposium on VLSI Circuits Digest of Technical Papers*. IEEE, 1996.

[17] A. Nalamalpu and W. Burleson. Repeater insertion in deep sub-micron cmos: Ramp-based analytical model and placement sensitivity analysis. In *IEEE International Symposium on Circuits and Systems*, 2000.

[18] J.-M. Parcerisa and A. Gonzalez. Reducing wire delay penalty through value prediction. In *International Symposium on Microarchitecture*, pages 317–326, 2000.

[19] Y. Sazeides and J. Smith. The Predictability of Data Values. In *MICRO*, 1997.

[20] Y. Shin, S.-I. Chae, and K. Choi. Partial bus-invert coding for power optimization of system level bus. In *Proceedings 1998 international symposium on Low power electronics and design*, pages 127–129. ACM Press, 1998.

[21] P. Sotiriadis and A. Chandrakasan. Low Power Bus Coding Techniques Considering Inter-wire Capacitances. In *IEEE Custom Integrated Circuits Conf.*, 2000.

[22] W. B. Srividya Srinivasaraghavan. Interconnect effort - a unification of repeater insertion and logical effort. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI'03), February 20 - 21, 2003*, Februray 2003.

[23] M. Stan and W. Burleson. Bus-Invert Coding for Low-Power I/O. *IEEE Trans. on VLSI*, 3(1):49–58, 1995.

[24] H. Yu, J. Lee, and J. Cho. A Fast VLSI Implementation of Sorting Algorithm for Standard Median Filters. In *IEEE Int'l ASIC/SOC Conf.*, 1999.

[25] H. Zhang, V. George, and J. Rabaey. Low-Swing On-Chip Signaling Techniques: Effectiveness and Robustness. *IEEE Trans. on VLSI*, 8(3):264–272, 2000.

[26] C. Zukowski and S. Wang. Use of Selective Precharge for Low-power Content-addressable Memories. In *IEEE Int'l Symp. on Circuits and Systems*, 1997.