3-2007

Lightweight Error Correction Coding for System-Level Interconnects

Jason D. Bakos
University of South Carolina - Columbia, jbakos@cse.sc.edu

Donald M. Chiarulli

Steven P. Levitan

Publication Info

Published in *IEEE Transactions on Computers*, Volume 56, Issue 3, 2007, pages 289-304.
http://ieeexplore.ieee.org/servlet/opac?punumber=12

Lightweight Error Correction Coding for System-Level Interconnects

Jason D. Bakos, *Member*, *IEEE*, Donald M. Chiarulli, *Member*, *IEEE*, and Steven P. Levitan, *Senior Member*, *IEEE*

Abstract—"Lightweight Hierarchical Error Control Coding (LHECC)" is a new class of nonlinear block codes that is designed to increase noise immunity and decrease error rate for high-performance chip-to-chip and on-chip interconnects. LHECC is designed such that its corresponding encoder and decoder logic may be tightly integrated into compact, high-speed, and low-latency I/O interfaces. LHECC operates over a new channel technology called Multi-Bit Differential Signaling (MBDS). MBDS channels utilize a physical-layer channel code called "N choose M (nCm)" encoding, where each channel is restricted to a symbol set such that half of the bits in each symbol are set to one. These symbol sets have properties that are utilized by LHECC to achieve error correction capability while requiring low or zero relative information overhead. In addition, these codes may be designed such that the latency and size of the corresponding decoders are tightly bounded. The effectiveness of these codes is demonstrated by modeling error behavior of MBDS interconnects over a range of transmission rates and noise characteristics.

Index Terms—Interconnections (subsystems), interconnection architectures, code design, coding tools and techniques, coding and information theory, error control codes.

1 Introduction

The semiconductor industry continues to improve fabrication technology by shrinking minimum feature size and oxide thickness, allowing for a continual increase in logic speed and integration density. The microprocessor industry sustains its growth in raw computational power by developing increasingly aggressive microarchitectures that are tailored to take advantage of these new speeds and densities. However, the relative performance of system-level interconnects has, for the most part, remained stagnant relative to the capabilities of the corresponding fabrication technology. In fact, advances in semiconductor fabrication technology have, in many ways, created new challenges for implementing correspondingly fast interconnect technologies. For example, decreases in supply voltage have narrowed signal margins, growth in integration density has increased relative levels of noise that affect the power supply and substrate, finer wire pitches have increased the effect of crosstalk of parallel signal conductors, increasing power requirements require relative increases in the portion of I/O pads reserved for the power supply (leading to I/O-constrained designs), and wire delay has become increasingly significant relative to clock frequency.

In this paper, we propose the use of error correction coding to increase the end-to-end throughput of off-chip interconnects. This technique exploits the new capacities for logic speed and integration to couple encoding and decoding logic to inter and inner-chip interfaces. Traditionally, error correction codes have not been a viable approach for such channels due to the high logic complexity and high latency required for decoding, preventing decoder integration into existing chips and requiring unacceptable latency. As a result, current research in high-performance short-haul channels has primarily focused on analog approaches (as opposed to information-theoretic approaches) such as signal preemphasis and equalization.

Our technique is designed to take advantage of a new channel technology which itself is an extension of current-mode differential signaling. Current-mode differential signaling technology offers robust noise immunity at the physical layer and has become an attractive solution for high-performance, serialized system-level interconnects. As such, the industry has adopted differential signaling for applications that require high-throughput off-chip interconnects. For example, differential signaling is used for such standards as LVDS [2], Xilinx's Multi-Gigabit Transceivers [9], the DVI digital display interface [10], and the physical-layer of network architectures such as HyperTransport and RapidIO [11]. There has also been work in adapting differential signaling to on-chip interconnects [3]. However, differential signaling is expensive because it requires two physical conductors for each bit of interconnect capacity, along with high static power consumption. In many cases, this limits the use of differential signaling to bit-serial interconnects. Multi-Bit Differential Signaling (MBDS) [1], [13], [14] has emerged as a more efficient alternative to differential signaling because it achieves equivalent noise immunity while requiring lower I/O and power resources. Due to the unique structure of the MBDS driver circuit and termination network, MBDS channels require that data transmitted across the channel be encoded with a physical-layer channel code called "N choose M

- *J.D. Bakos is with the Department of Computer Science and Engineering, University of South Carolina, Columbia, SC 29208. E-mail: jbakos@cse.sc.edu.*
- *D.M. Chiarulli is with the Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260.*
- *S.P. Levitan is with the Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA 15261.*

Manuscript received 23 Nov. 2005; revised 10 May 2006; accepted 10 Aug. 2006; published online 22 Jan. 2007.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0414-1105.

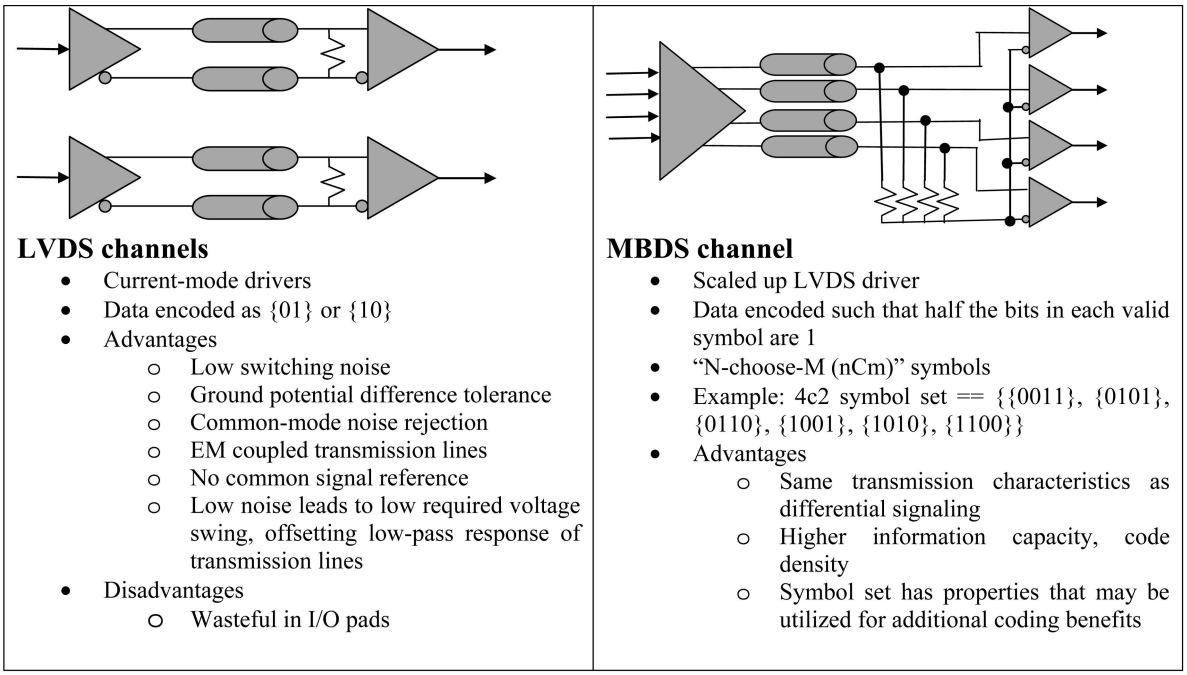

Fig. 1. Side-by-side comparison of LVDS and MBDS channels.

(nCm)" encoding.[1] In an nCm physical-layer code, data is restricted to a predefined binary symbol set where a channel of arbitrary physical width carries symbols where half the bits are binary-1 values. This physical code contains inherent data redundancy that can be exploited to gain additional encoding benefits.

This paper describes Lightweight Hierarchical Error Control Coding (LHECC). LHECC endows MBDS-based interconnects with error correction capability while requiring low information and implementation overhead. LHECC is based on nonlinear block codes and relies on a hierarchical approach to combine both symbolic and binary error control encoding techniques. LHECC takes advantage of existing information redundancy inherent within the underlying channel in order to reduce or eliminate the relative amount of information overhead required to achieve error correction capability. In addition, LHECC requires low logic complexity through its use of small block lengths and lookup table-based encoding and decoding. Also, the hierarchical nature of the encoding and decoding procedure naturally lends itself to a shallow pipelined approach for hardware implementation. These properties allow these codes to have minimal encoding and decoding area overhead while simultaneously exhibiting error correction capability, low end-to-end latency, and high end-to-end throughput.

The remainder of this paper is organized as follows: Section 2 provides a short overview of MBDS channel technology. Section 3 describes LHECC and how these codes can be implemented with compact encoders and decoders at system-level interfaces. Section 4 describes how the effectiveness of LHECC is measured through modeling off-chip MBDS channels. Section 5 lists experimental results and Section 6 concludes the paper.

1. Also referred to as "m-out-of-n" encoding in prior work.

2 Multi-Bit Differential Signaling (MBDS)

Multi-Bit Differential Signaling (MBDS) [1], [13], [14] is an alternative to low-voltage differential signaling (LVDS) for building system-level interconnects. MBDS channels retain the favorable noise immunity and transmission characteristics of differential channels while exhibiting higher information density. This allows MBDS channels to make more efficient use of I/O pad and power resources. In an MBDS channel, a standard current-mode low voltage differential (LVDS) driver design is scaled up such that it drives more than two wired outputs. At the receiver, all wires are match-terminated into a star termination network. The center of the star network, referred to as the common point, becomes a signal reference for standard LVDS receiver circuits. By restricting the output of the driver such that exactly half of the outputs are sourcing current (leaving the other half as return paths), the voltage of the common point is held constant at the DC offset of the driver.

This technique requires a physical-layer channel code, called "N choose M (nCm)" encoding. An MBDS channel is built with an arbitrary width n, where every symbol contains m one-bits, where $m = \text{floor}(n/2)$, and may transmit any of $n!/((n-m)!m!)$ valid symbols. One differential receiver is used for each of the wires forming the channel. As in an LVDS channel, a positive or negative voltage across each receiver's corresponding termination resistor (relative to the common point) is interpreted as a binary value. A comparison of LVDS and MBDS channels is shown in Fig. 1. For example, an 8-wire wide MBDS channel sources four units of current and may transmit any of 70 valid symbols, yielding a capacity of 6 bits. A differential link with equivalent capacity would require 12 wires and source six units of current, for a savings (MBDS over LVDS) of 33 percent in both energy and wire count.

3 Lightweight Hierarchical Error Control Codes

Lightweight Hierarchical Error Control Codes are designed as a link-layer code that is applied over the physical-layer nCm code used for MBDS channels. In addition to providing forward error correction, LHECC defines a technique for encoding binary data for interconnects formed from multiple parallel MBDS channels. LHECC utilizes a hierarchical approach to encoding source data that allows it to make use of the inherent redundancy within nCm symbol sets. This serves to minimize both the code overhead and decoding complexity/latency required to endow MBDS-based interconnects with error correction capability. This section describes LHECC, how the codes are encoded and decoded, and an example hardware implementation.

3.1 Hierarchical Encoding and Decoding

To construct a code word, binary source data is logically split into two portions referred to as the *s-data* and *c-data*. The s-data portion is encoded using the *high-level code*, consisting of a traditional (n, k, d, q) linear block code.[2] When encoding the high-level code, one or more parity symbols are computed and appended to the s-data. This encoding forms the high-level code block. The low-level code is used to combine the high-level code block with the c-data to select the actual sequence of nCm symbols that is used to form the code word, which is then used for transmission across an interconnect formed with a small number of parallel MBDS channels. The resulting nonlinear code space has an arbitrary distance that is set by the code designer. Most importantly, it achieves error control without requiring dedicated parity symbols that do not carry any source data. In addition, fewer parity symbols are required to correct symbols within the code word as compared to a traditional block code because most errors manifest themselves as invalid nCm symbols. Invalid symbols may be detected at the receiver as symbol erasures instead of symbol errors. Together, these attributes make the code lightweight.

3.2 Low-Level Code

By definition, every nCm symbol set has an inherent ability to detect certain types of bit errors that affect a symbol during transmission over a symmetric channel. An nCm symbol containing bit errors will be detected when the symbol sampled by the receiver does not contain an appropriate number of one-bits as defined by half the symbol width. As a result, any random occurrence of bit flips occurring within an nCm symbol will be detected as an error by the receiver unless an equal number of 0-bits and 1-bits in the original symbol are changed as a result of the errors. This implies that any transmission errors that are manifested as an odd number of bit errors within a symbol can be detected, while only a subset of transmission errors that are manifested as an even number of bit errors within a symbol can be detected.

TABLE 1
Example Partitioning of a 6c3 Symbol Set Into Four Subsets of Four Symbols Each, Using Target Hamming Distance of 4

subset base-4	**symbols base-4**
0	000111, 011100, 101010, 110001
1	001011, 010110, 100101, 111000
2	001101, 011010, 100011, 110100
3	001110, 010101, 101001, 110010

The low-level code relies on transforming an nCm symbol set in such a way that its inherent error *detecting* capability is increased to become an error *correcting* capability. Observe that any pair of randomly selected nCm symbols chosen from any given nCm symbol set will differ in at least two bit positions and, thus, the effective Hamming distance is 2. The number of detectable errors for any code set is defined as $d-1$, where d is the distance. However, the number of correctable bit errors is defined as $\mathrm{floor}((d-1)/2)$. This means that a code set must have a distance of no less than 3 before it gains the capability to correct errors. As such, the Hamming distance of an nCm symbol set must be increased in order to achieve capability for error correction. This can be accomplished by partitioning the nCm symbol set into subsets such that the distance of each subset is defined by the distance parameter of the partitioning operation. This distance may be any even value, ranging from 4 to the width of the symbol. For example, partitioning for distance 4 would allow correction of any single-bit error affecting any nCm symbol within a particular subset, while partitioning for distance 6 would allow correction of any 1 or 2-bit errors affecting any nCm symbol within a particular subset. The only restriction of this partitioning operation is that each of the resulting subsets must be of equal size. The partitioning operation forms the low-level code and is defined as an (nCm, s, c, d_l)-code, where nCm defines the nCm symbol set that is partitioned, s defines the number of subsets, c defines the number of symbols per subset, and d_l defines the Hamming distance of each subset. The subsets are enumerated such that each subset is assigned a unique base-s value. Likewise, the nCm symbols within each subset are enumerated such that each symbol is assigned a unique base-c value that identifies the symbol within its corresponding subset. This enumeration allows every nCm symbol to be associated with a corresponding subset value, or s-data value, and symbol value, or c-data value. An example partitioning of a 6c3 symbol set into four subsets of four symbols each with a target distance of 4 is shown in Table 1.[3]

2. n defines the number of symbols that form a code block (the basic unit over which error control is performed), k defines the number of data symbols encapsulated within the block, d defines the Hamming distance of resultant code space, and q defines the base of each of the symbols that form the block (which, together with k, defines the actual data capacity of each symbol).

3. Note that this example partitioning only assigns 16 out of the 20 possible symbols in the 6c3 symbol set (four symbols are not used). As is the case with many nCm symbol sets, it is impossible to partition the 6c3 symbol set with $\mathrm{distance} = 4$ if all symbols must be assigned to a subset.

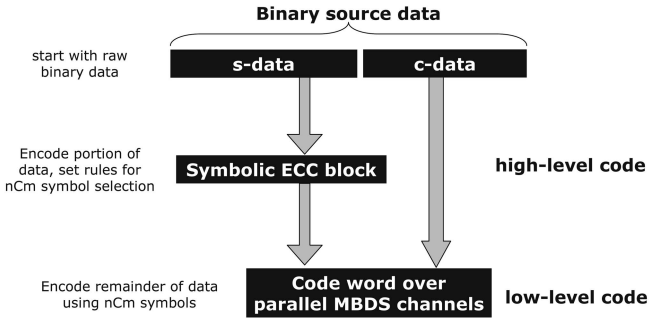

Fig. 2. Hierarchical ECC encoding.

3.3 High-Level Code

The high-level code of LHECC is a traditional (n, k, d_h, q) linear block code. The error detection and correction capability of this code depends solely on its distance. However, the power contributed from the low-level code guarantees that the overall error correction power of the LHECC is higher than the error correction power of the high-level code alone. A block code can detect up to $d_h - 1$ symbol errors and correct these errors if the receiver has knowledge of which symbols within the block are in error. These types of errors are called *erasures*. The block code can correct $\text{floor}((d_h - 1)/2)$ symbol errors in the block when the receiver does not have a priori knowledge of which of the symbols in the block were received in error.

In order to design an LHECC-encoded interconnect, the designer must decide on several parameters that ultimately determine the interconnect's overall physical width, data capacity, and error correcting power. These parameters are the MBDS channel width, number of MBDS channels, maximum number of correctable bits within each individual nCm symbol, and maximum number of correctable nCm symbols within the code word. The MBDS channel width defines the nCm symbol set that is used for the partitioning operation. This width, when multiplied by the number of MBDS channels used to form the interconnect (high-level block size n), defines the actual width (in wires) of the interconnect.

The target distance for the partitioning operation determines the maximum number of correctable bits within each nCm symbol. Together, the maximum number of correctable bits within each symbol and the maximum number of correctable nCm symbols determine the required distance of the high-level block code. The code is capable of correcting up to $d_h - 1$ nCm symbol errors, assuming these symbols are received as erasures (invalid nCm symbols) and each of these symbols contain less than or equal to $\text{floor}((d_l - 1)/2)$ bit errors. Likewise, the LHECC is capable of correcting up to $\text{floor}((d_h - 1)/2)$ nCm symbols errors, assuming these symbols are received as pure errors (incorrectly received but valid nCm symbols) and each of these symbols contain less than or equal to $\text{floor}((d_l - 1)/2)$ bit errors. Given the parameters of the high-level and low-level codes, an LHECC configuration is formally defined as an $(nCm, n, k, d_h, s, c, d_l)$ code.

TABLE 2
Partitioning for 4c2 Symbol Set

	c	
s	0	1
0	0011	1100
1	0101	1010
2	0110	1001

3.4 Encoding

Fig. 2 shows an overview of the encoding procedure. Segments of source data are split into an s-data portion and a c-data portion. The s-data portion, represented as a sequence of base-s symbols, is encoded using the high-level block code. The c-data portion, represented as a sequence of $\text{base-}c$ symbols, is encoded by choosing the nCm symbols corresponding to each of the c-data values from each of the subsets specified in the sequence of values that make up the high-level code block (s-data symbols and corresponding $\text{base-}s$ parity symbols).

For example, assume a simple interconnect formed with three parallel 4c2 channels. Assume the 4c2 symbol set is partitioned into three subsets, two symbols per subset, and subset distance of 4, as shown in Table 2.

Assume a high-level code consisting of a $(n = 3, k = 2, d = 2, q = 3)$ checksum code. This high-level code is capable of correcting one erasure. This code carries 3 bits into its s-data and 3 bits in its c-data, yielding an interconnect capacity of 6 bits using 12 wires. Assume the transmitter wishes to encode and transmit the binary sequence "111101_2" across the three 4c2 channels. The following actions occur at the encoder:

- The most significant 3 bits become the s-data. 111_2 is converted from base-2 to base-3, yielding an s-data representation of 21_3.
- A checksum is computed and appended to the s-data, forming the high-level code block $2 + 1 \pmod 3 = 0_3$.
- 210_3 becomes the high-level code block. The least significant 3 bits of the source data (the c-data) is used to choose an nCm symbol from the subsets specified by the high-level code block. Using Table 2, 210_3 is combined, digit by digit, with 101_2 to form the code word of $1001\ 0101\ 1100_2$.
- The code word is used as an input to three parallel 4c2 MBDS drivers.

3.5 Decoding

The receiver decodes the code word by following a discrete sequence of steps. First, each nCm symbol making up the code word is "resolved" into its corresponding s-data value, c-data value, and an error flag. The error flag indicates if the symbol was received as an invalid nCm symbol. Symbols resolved with an error flag are treated as erasures relative to the high-level code. Next, the decoder performs a preliminary check to determine if the code word was received with too many errors to be decoded. There are two possible ways this can occur. First, the decoder determines if there are more erasures in the code word (signaled by error flags) than the high-level block code can correct. Next, the high-level code

TABLE 3
nCm Symbol Set Partitionings Achieved through a Branch-and-Bound Search Algorithm

symbol set	symbols	distance	subsets (s)	symbols/subset (c)	s/c	overhead/parity	relative overhead
4c2	6	4	3	2	1.50	1.5	.79
6c3	20	6	10	2	5.00	3.3	.83
6c3	20	4	4	4	1.00	2	.50
8c4	70	4	10	7	1.43	3.3	.55
8c4	70	8	35	2	17.50	5.1	.85
8c4	70	4	7	9	0.78	2.8	.47
8c4	70	4	8	8	1.00	3	.50
10c5	252	4	8	16	0.50	3	.43
10c5	252	6	32	6	5.33	5	.71
12c6	924	4	8	64	0.13	3	.33

block is checked to ensure that it doesn't contain more symbol errors than can be corrected by the high-level code. This can occur if symbol errors exist that aren't detected as invalid nCm symbols and the high-level code block contains more errors than can be corrected by the high-level code.

When errors occur with a code word, each of the high-level code block symbols that correspond to each of the nCm symbols received in error are corrected in parallel using the high-level code. This allows the successful extraction of the entire high-level code block, the corresponding subset values, and the s-data. These corrected high-level code symbols, along with each corresponding sampled nCm symbol received in error, are used to compute the corrected nCm symbols from the code word. This is performed by determining which nCm symbol within the subset specified by the corrected high-level code block symbol value has minimum distance relative to the received symbol. Next, the c-data is determined from the received and corrected nCm symbols. Finally, the s-data and c-data are converted to base-2 (if required), yielding the original binary source data.

For example, assume the code word from the previous example, $1001\ 0101\ 1100_2$, is transmitted across the interconnect and noise affecting the channels causes the receiver to sample the following code word: $1101\ 0101\ 1100_2$. Note that the most significant nCm symbol contains one bit error. The following actions occur at the decoder:

- The most significant symbol contains three 1-bits, meaning that it is an invalid nCm symbol.
- The code word is resolved to a high-level code block of $?10_3$. Checksum codes have the ability to correct one erasure, so combinational logic in the decoder computes $0 - 1 \pmod 3 = 2_3$.
- The symbol received in error belongs to subset 2. A minimum distance decoder is used to determine which of the symbols in subset 2 most closely resembles 1101_2.
- Of the two symbols in subset 2, 0110_2 differs from the received symbol in three bit positions, while 1001_2 differs in only one bit position. The symbol is corrected to 1001_2.

3.6 LHECC Overhead

Overhead is defined as the effective number of information bits carried by the code word that are effectively lost due to the code redundancy required for forward error correction. Recall that the only components of a code word that are strictly used for redundancy are the parity symbols of the high-level code block. In other words, instead of relegating entire nCm symbols in the code word to be used as parity information, a portion of the c-data is effectively "piggy-backed" onto the parity symbols of the high-level code. However, relative to the high-level code block, fewer parity symbols are required to achieve error correction capability as compared to a traditional block code. This is because bit errors that manifest themselves as invalid symbols can be corrected as erasures relative to the high-level code and fewer parity symbols are required to correct erasures as opposed to errors. In this case, overhead is defined as:

$$Overhead_{bits} = \lfloor (n-k) \bullet \log_2 s \rfloor.$$

The base of the c-data determines how much source data is carried within the parity symbols in the high-level code block. Therefore, overhead is minimal when the value of s is minimized while the value of c is maximized. This is an important consideration for partitioning nCm symbol sets. Another way to measure information efficiency for an interconnect with LHECC is to determine its code rate relative to a raw interconnect (without LHECC). We refer to this as relative code rate, defined as the bit capacity of an interconnect with LHECC divided by the bit capacity of the same interconnect without LHECC (assuming independently binary-mapped parallel nCm channels). This measurement is defined as:

$$Rate_{relative} = \frac{\lfloor k \bullet \log_2 s \rfloor + \lfloor n \bullet \log_2 c \rfloor}{n \bullet \lfloor \log_2 nCm_{size} \rfloor}.$$

Table 3 lists possible partitionings of several nCm symbol sets. Each of these partitionings was achieved through the use of a branch-and-bound search over the space of symbol-subset assignment states. The column labeled "overhead/parity" expresses the overhead required for each high-level parity symbol in terms of bits, while the

column labeled "relative overhead" expresses the overhead for each high-level parity symbol in terms of percentage of a single symbol's capacity. As shown in this table, given a symbol set and target distance, overhead is minimal when the number of subsets (s) is minimized relative to the number of symbols per subset (c). The next section will describe how this "rule of thumb" also serves to reduce the complexity of the decoder architecture.

3.7 Encoder/Decoder Architectures

This section discusses implementation details regarding encoder and decoder architectures for LHECC-encoded interconnects. Such circuits must be both compact (allowing integration into system-level I/O interfaces) as well as capable of operating at high operating frequency. LHECC has three properties that make this possible. First, due to the low relative code overhead of LHECC, the encoders and decoders may be designed for codes with small block lengths of three to four symbols while still retaining a high relative code rate. Small block sizes serve to reduce overall decoding complexity because the core operations within the encoder and decoder may be based on small, low-latency combinational logic circuits (implementing lookup tables). Second, the partitioning operation effectively reduces block code decoding complexity because it ensures that the symbol base of the high-level code is reduced relative to the base of the nCm symbol set. In other words, decoding may be logically split in two discrete steps, where each step operates over a symbol base less then the native symbol base of the nCm symbols. Third, the hierarchical nature of LHECC allows for a clear division of tasks for parallel (pipelined) implementation. Note that the pipelined design of the encoder and decoder described in this section add end-to-end latency to the interconnect. *In this work, we strictly regard end-to-end throughput as a design goal that carries higher priority than end-to-latency.*

3.7.1 Example Encoder/Decoder Architecture

We utilize a sample interconnect as a backdrop to discuss a possible encoder and decoder implementation. This sample interconnect consists of 18 wires that encode 10 binary bits, configured as three 6c3 channels. Each circuit described in this section is synthesized HDL targeting a simple standard cell library consisting of six, minimal-sized primitive logic cells: 1-bit latch, inverter, 2 and 3-input NAND gate, and 2 and 3-input NOR gate. Our sample interconnect consists of three 6c3 channels, where the 6c3 symbol set is partitioned into four subsets of four symbols per subset and a high-level block code consisting of an $(n = 3, k = 2, d = 2, q = 4)$-checksum. This code can correct one bit error affecting one nCm symbol per code word.

As is common in code theory, the encoder implementation is relatively simple as compared to the decoder implementation. An example encoder is shown in Fig. 3. This encoder is divided into two pipeline stages and requires 208 gates and 30 latches. The "high-level code" stage of the encoder is where the checksum parity symbol is computed from the two base-4 s-data values. This component is based on table lookup and requires 19 gates and five gate delays. Slightly more sophisticated codes can also be

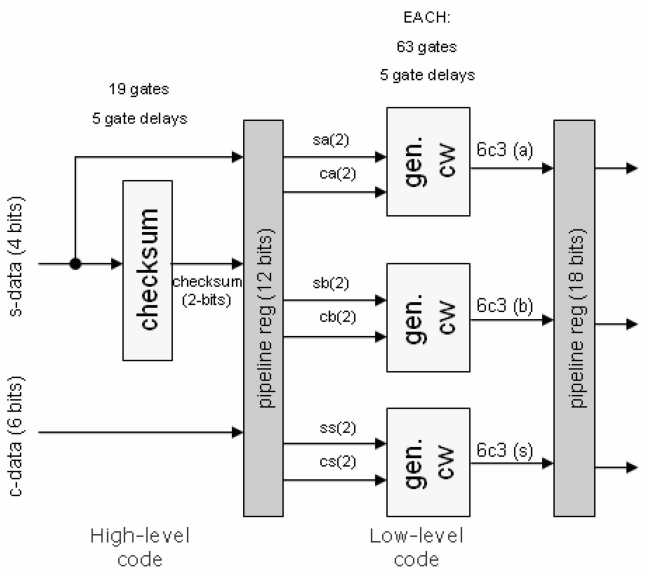

Fig. 3. Example encoder architecture.

encoded in only one stage with a similar number of logic gates. For example, an $(n = 4, k = 2, d = 3, q = 3)$ maximum distance separable (MDS) code, requiring generation of two base-3 parity symbols, can be built with 37 gates and five gate delays. The "low-level" stage of the encoder is responsible for generating inputs to the MBDS drivers based on the high-level code block and the c-data values for each symbol in the code word. In this case, each of these components requires 63 gates with five gate delays (in parallel) each. The final pipeline register is used to ensure the inputs to each of the MBDS drivers remains stable for a full clock cycle.

An example decoder corresponding to the encoder shown in Fig. 3 is shown in Fig. 4. This decoder is divided into six pipeline stages and requires 830 gates and 128 latches. The first pipeline register is responsible for sampling and latching the symbols from the MBDS receivers. In the "resolver" stage, each of the symbols is resolved into its corresponding s-data value, c-data value, and error flag. The resolver components require 130 gates each and seven gate delays (in parallel). The "high-level code" stage is responsible for managing the high-level code. This involves performing preliminary checks for nondecodable code words and high-level symbol correction. The "block error detector" determines if there are too many errors in the code word to correct. This occurs when there are too many erasures (error flags) or when there are no detected erasures but the high-level code block is invalid. This component requires 47 gates and seven gate delays. The "block corrector" component, operating in parallel, is responsible for determining the subset value of any single nCm symbol that is received as an invalid nCm symbol. This component requires 49 gates and seven gate delays. Since these operations are performed in parallel, the entire stage requires seven gate delays.

More sophisticated high-level codes may also be implemented in one pipeline stage. For example, for an $(n = 4, k = 2, d = 3, q = 3)$ block code (capable of correcting up to two erasures or one pure error), the block code error detector requires 106 gates and seven gate delays and the

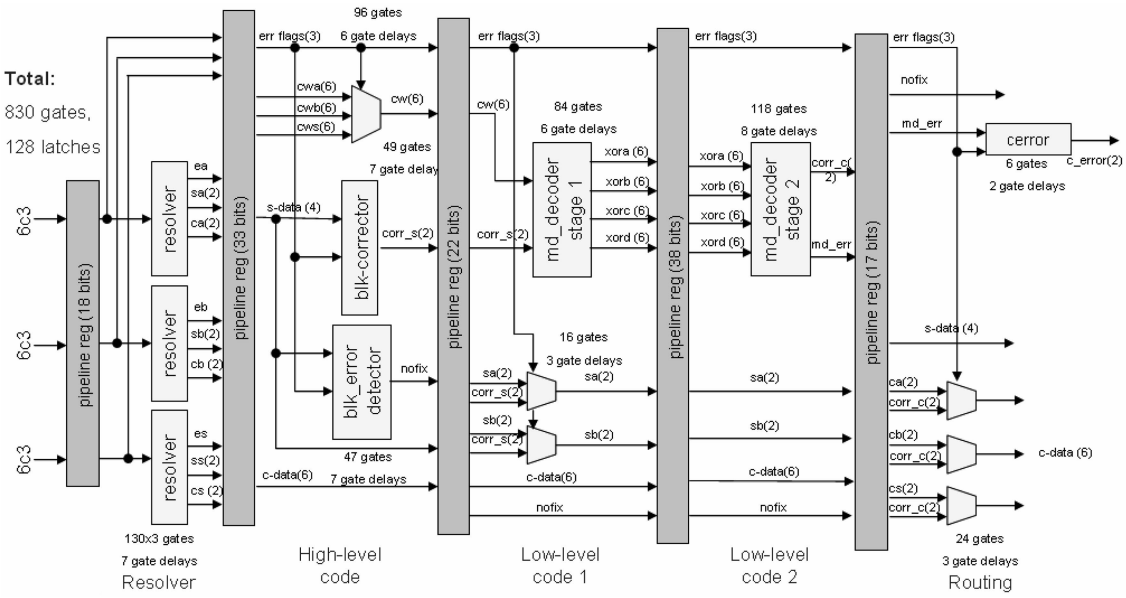

Fig. 4. Example decoder architecture.

block code error corrector was designed such that it requires 364 gates and seven gate delays.

Referring again to the example decoder shown in Fig. 4, "Low-level code 1" and "Low level code 2" are two stages that manage the low-level code. They perform minimum distance decoding based on the subset value computed by the block corrector and the received symbol. This operation is split into two stages in order to reduce the per-stage latency requirements for minimum distance decoding. If a received nCm symbol is equidistant to two valid nCm symbols in its subset, a flag is raised, indicating that the errors cannot be corrected. Together, these stages require 202 gates and require six gate delays in the first stage and eight gate delays in the second stage. Finally, the "routing" stage prepares the decoded data for the on-chip reception and distribution.

The encoder and decoder designs described in this section are based on combinational logic-based lookup tables as opposed to arithmetic operations. Unfortunately, this prevents these implementations from being scalable relative to symbol and block width. This is a disadvantage because code density increases with symbol width and block width. However, scalability is not an important goal in this work for two reasons. First, next-generation system-level interconnects are expected to become serialized (narrow) in order to achieve low I/O and power resources. This means that wide interconnects (resulting from the use of multiple parallel wide nCm channels) are not favorable.[4] Second, short symbol and block widths facilitate shallow pipeline stages and thus allow for decoders that are simple, low latency, and capable of operating at high pipeline speed. This allows the code/interconnect designer to place tight bounds on the size and latency of each lookup table required by the decoder. Although traditional error correction codes become inefficient (exhibit low code rate) when used with short symbol and block widths, LHECC sidesteps this problem by virtue of having lower overall relative code overhead.

Table 4 describes how each major component in the above decoder implementation scales in complexity (latency and number of logic gates) relative to symbol width, s, c, block width, and number of parity symbols. As shown in the table, several components have exponential growth as certain parameters are scaled. However, the example implementation illustrates how small block sizes and symbol widths can be reasonably implemented with less than 1,000 total gates (trivial by today's standards when 10s of millions of gates are typical for large-scale digital systems) and the experimental results illustrate a substantial benefit in signal integrity for these parameters.

Table 5 lists the gate count, maximum pipeline stage latency, and pipeline depth of several additional sample encoder and decoder implementations. Each implementation requires on the order of 1,000 gates, which constitutes a trivial requirement for both real estate area (as compared to typical logic integration densities) and power consumption (as compared to the power consumed by the corresponding current-mode channels). We also note that, while more aggressive approaches to designing the decoder logic may exist, such work is outside the scope of this paper.

3.7.2 Decoder Latency

Note that the decoder pipeline shown in Fig. 4 has a maximum pipeline stage latency of seven primitive logic gate delays[5] and a pipeline depth of 5. Note that the per-wire signaling speed of the channels afforded by future chip and PCB fabrication technologies may require a deepening of the decoder pipeline in order to match the speed of the corresponding interconnect. If this is the case, the pipeline may be deepened by splitting the logic of individual components over multiple pipeline stages at the cost of additional pipeline registers and additional end-to-end

4. One obvious solution to this would be to use wide nCm symbols but only one or two channels (trade off symbol width with block width). However, this would actually *decrease* overall code density because the relative overhead required by the high-level code's parity symbols would cause a dramatic increase in overall code overhead.

5. Roughly equivalent to seven FO4 gate delays.

TABLE 4
Scalability of Decoder Components

component	function	w (sym width)	s	c	n (block width)	n-k (parity sym)
resolver	maps each nCm symbol to s-value, c-value, and error flag	$O(2^{w-2})$ **exponential**	$O(1)$ **no effect**	$O(1)$ **no effect**	$O(n)$ **linear**	$O(1)$ **no effect**
block error corrector	corrects s-values caused by erasures and errors in the high-level code block	n/a	$O(s^n)$ **quadratic**	$O(1)$ **no effect**	$O(s^n)$ **exponential**	$O(s^{\lfloor (n-k)/2 \rfloor})$
block error detector	determines if there exists an uncorrectable number of errors in the high-level code block	n/a	$O(s^n)$ **quadratic**	$O(1)$ **no effect**	$O(s^n)$ **exponential**	$O(s^k)$
minimum distance decoder	corrects nCm symbol as a function of s-data value and received symbol, determines if there exists more nCm symbol errors than can be corrected	$O(2^{w-2})$ **exponential**	$O(sc\lfloor (d_f-1)/2 \rfloor)$ **linear**	$O(sc\lfloor (d_f-1)/2 \rfloor)$ **linear**	$O(n)$ **linear**	$O(n)$ **linear**

TABLE 5
Statistics for Example LHECC Architectures

Interconnect	Width	Bits	Correction capability	Encoder			Decoder			Max latency/stage
				Gates	Latches	Stages	Gates	Latches	Stages	
3x4c2	12	6	1 bit in 1 sym	100	28	3	328	66	5	7
4x4c2	16	7	1 bit in 2 sym	144	36	3	804	124	6	8
3x6c3	18	10	1 bit in 1 sym	208	30	2	830	128	6	8
3x8c4	24	15	1 bit in 1 sym	424	44	2	1246	234	6	8

latency. This is demonstrated by the decoder shown in Fig. 4, where the "minimum distance decoder" behavior is split into two stages by feeding intermediate results from the first pipeline stage into a second stage.

We have intentionally targeted a standard digital CMOS fabrication process (as opposed to a specialized high-performance RF analog process) when designing the MBDS drivers and receivers. This decision is based on our design requirement to allow integration of the drivers and receivers as drop-in IP components into existing high-speed digital designs such as microprocessors and FPGAs. Therefore, because the drivers and receivers are designed using standard digital MOSFET devices, their maximum signaling speed is expected to scale at the same rate as the digital logic speed of their corresponding decoders. Advances in PCB dielectric material notwithstanding, the decoder pipeline depth is nominally fixed and independent of the chip fabrication technology and channel speed. As a result, the relative speed of the decoder and the associated interconnect will scale with fabrication technology without the need for deepening the pipeline.

4 Experimental Design

We measure the effectiveness of LHECC by modeling chip-to-chip interconnects consisting of three to four parallel MBDS channels. These models capture error behavior over a range of transmission rates and noise characteristics. As shown in Fig. 5, an MBDS channel of width n consists of a model for a packaged transmitter chip, packaged receiver chip, PCB transmission line models, and a termination network model.

The interconnect models are simulated in the time-domain using process-defined CMOS voltage levels to inject a pseudorandom sequence of valid code words into the drivers and interpreting the corresponding outputs from the receivers after the latency of the channel (using process-defined voltage ranges).

4.1 Channel Model

Included in our channel model are the effects of the driver and receiver devices that form the end-points of the channel. These devices themselves are responsible for generating certain types of noise that affect the performance of the channel. In addition, these devices have their own limitations for transmitting and interpreting the signal

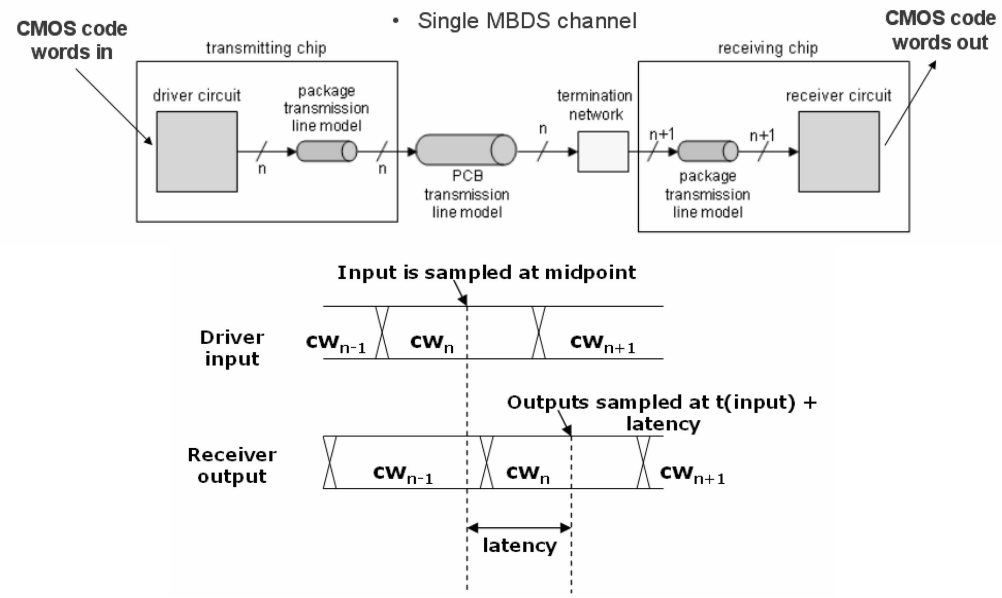

Fig. 5. PCB-based MBDS channel model.

carried by the wires that form the interconnect (based on the characteristics of the fabrication technology). The driver and receiver circuits are based on MOSFET transistor models from the Taiwan Semiconductor Manufacturing Corporation (TSMC) .18 um commercial design kit.

As shown in Fig. 6, the driver circuit is a current-steering design that supports an arbitrary channel width. The transistors in each driver are sized and biased such that each driver output provides a mean-absolute-value current of 7.5 mA.[6] As shown in Fig. 7, the receiver circuit is a self-biasing 2-stage LVDS receiver from the literature [4]. The receiver contains concurrent N-type and P-type differential stages, allowing for a wide common-mode input range. The output of the receiver is amplified and conditioned to CMOS levels using two tapered CMOS inverters in series. The receiver's role is to periodically sample and discriminate a differential input signal to determine if the current state of the signal represents a one-bit or zero-bit. In this sense, the receiver acts as an analog-to-digital converter and amplifier.

Each electrical connection from the driver and receiver circuits to the PCB is modeled as a wire bonded package connection. This connection consists of a wire bond model connecting the chip's I/O pad to a corresponding package I/O pad, with solder bump models at both ends. These models were generated by a package modeling tool from Chatoyant [5]. The model parameters are generated using realistic geometries and pitch for the chip, package, and wire bond features.

The channel model includes fully coupled transmission line models for 7.5″ PCB traces. PCB transmission lines exhibit frequency-dependent signal attenuation (low-pass frequency response) and crosstalk (AC coupling) among nearby traces. The traces are spaced and sized to match impedance to the termination network (50 ohms). Layer spacing distances are based on a standard PCB fabrication process and assume a dielectric material of FR-4. Models for these structures are generated with Cadence's Transmission Line Model Generator (LMG) [6]. This tool computes and generates lumped transmission line models. Each lump includes ground capacitance, serial inductance, and mutual capacitance/inductance from each wire to all other wires.

The channel model itself captures the degradation of the signal between the driver and receiver as a function of transmission frequency and external noise sources. This degradation occurs as a result of factors such as transistor switching noise, edge jitter, and frequency-dependant signal attenuation and crosstalk. The output of the receiver is a conditioned and amplified CMOS voltage that is suitable for on-chip transmission and use. However, during sampling, if the receiver's input signal is too badly

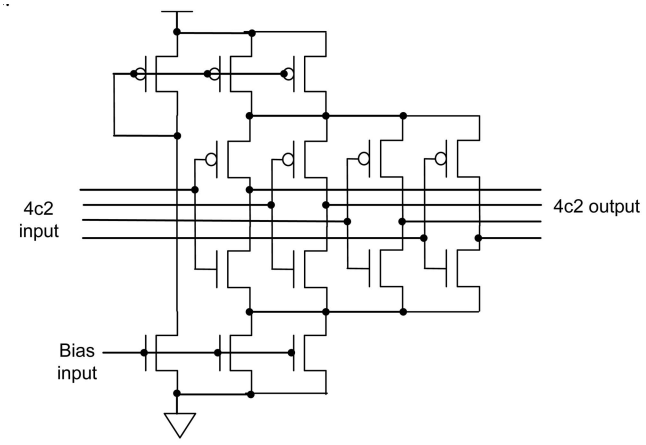

Fig. 6. Example 4c2 MBDS driver.

6. This value was chosen arbitrarily to keep the driver output magnitude consistent across each of the driver designs. This is also a reasonable value when compared to state-of-art systems. To clarify, this implies that an nCm driver would always be sourcing $(n/2)^{*}7.5$ mA of current and dissipating this current across a 50-ohm termination at the receiver (as well as a 50-ohm matched impedance fully coupled transmission line).

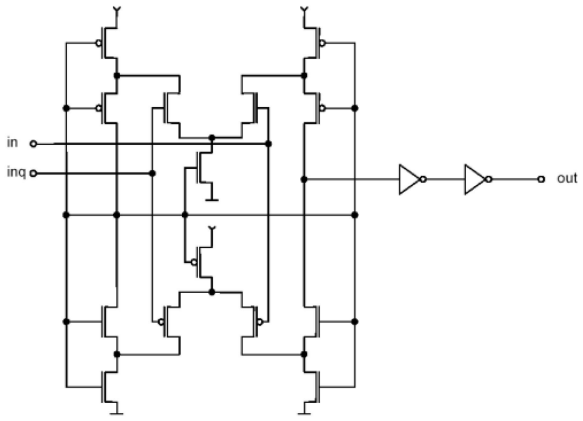

Fig. 7. Differential receiver design [4].

degraded or skewed, the receiver will incorrectly discriminate this signal at the sample time, resulting in a bit error.

Eye diagrams are used to illustrate this phenomenon. Each eye diagram is a plot of a signal continuously plotted over itself for two bit periods. These signals originate from a random bit stream. This plot resembles an eye and signal integrity is a function of the "openness" of the eye. In an eye diagram, jitter is shown by the thickening of the left and right sides of the eye, while noise is shown by the thickening of the top and bottom of the eye. Therefore, as the signal degrades, the eye closes. Eye diagrams over increasing transmission rates are shown in Figs. 8 and 9. Fig. 8 shows eye diagrams of the receiver input signal and Fig. 9 shows the corresponding receiver output signal. These eye diagrams demonstrate how increasing signal degradation across the channel causes higher numbers of bit errors at the receiver output. These errors are shown at the output of the receiver as increasing jitter and eventual signal breakdown. At high transmission rates, bit errors are visible at the receiver output.

4.2 Supply Noise Model

MBDS drivers and receivers are intended for use as a "drop-in" system interface for applications such as microprocessor system buses and network-on-chip channels (global on-chip interconnects). The switching activity of logic located elsewhere on the host chip generates noise on the chip's power supply. Supply noise is an increasingly important consideration for large-scale, high-speed digital systems [7]. Such noise would certainly affect system-level interfaces as fluctuation in supply voltage reduces signal margins for drivers and receivers. LHECC provides a method for recovering from errors caused by this type of noise. However, this type of noise is not inherently modeled by the channel model. In order to model this type of noise within the MBDS channel model, supply noise must be artificially generated and independently added as an external source to the supply voltage of both the driver and the receiver circuits.

Intel has measured the power supply noise on a busy Pentium 4 processor operating with a 1.5 V supply voltage [7], [8] by probing V_{dd} and V_{ss} pin pairs at various processor package locations. The voltage reading indicated a Gaussian signal centered at 1.5 V with a 17-20 mV standard deviation. These measurements provide a rough statistical approximation of supply noise characteristics for a typical microprocessor. Since this noise is generated from switching activity originating from CMOS logic, the noise is assumed to exhibit frequency characteristics centered on the core operating frequency (clock frequency) of the digital circuitry. When modeling this type of noise, it is assumed that the MBDS channels are operating at the same frequency as the core logic. To generate the noise signal, white Gaussian noise is generated for 20 μs at a sampling rate of 20 GHz. The noise is then filtered through a pass-band filter having a band of 200 MHz centered on the per-wire switching frequency of the channel that is being simulated. The standard deviation of the unfiltered Gaussian noise signal is set such that the standard deviation of the filtered signal is 30-40 mV.[7]

4.3 Fringe Capacitance

Other noise sources that are not inherently modeled by the channel model are effects from parasitic fringe capacitance that originate from the physical layout of the circuits and package. Interconnect formed by parallel wires is traditionally laid out on-chip as a group of adjacent traces. Likewise, the corresponding package I/O pads for these wires are also adjacent. The proximities of these features add additional crosstalk to signals within the channel. This crosstalk is modeled by fringe capacitance between adjacent signals within each channel. For the channel model, the value of these capacitors is 500 fF, verified as being realistic by the Cadence Diva extractor as reasonable fringe capacitance values.[8]

4.4 Experimental Interconnects

Section 5 presents results obtained through simulation of five different interconnect configurations. Each interconnect model is simulated with Cadence's Spectre device-level analog circuit simulation tool [6] (equivalent to SPICE). The simulations are designed to measure the relative amount of noise immunity gained through the addition of LHECC to raw MBDS-based interconnects. Each interconnect is associated with a code configuration having various levels of overhead and error correcting power.

The results are collected by counting the number of code word errors that would occur at the receiver given the presence and absence of LHECC over a simulation run of 100,000 code word transmissions. A code word error occurs when signal errors prevent encoded source data from being successfully decoded back to its original source data. In the case of an interconnect without LHECC, a code word error occurs whenever any single bit within any of the nCm symbols carried by the interconnect is received in error. In the case of an interconnect with LHECC, a code word error occurs whenever there are more bit errors present within the received code word than the code is capable of correcting.

7. The standard deviation of our generated noise is purposely set higher than the measured standard deviation on the Intel processor in order to compensate for the higher supply voltage of the TSMC .18um technology (1.5 V to 1.8 V), as well as to slightly exaggerate the noise to test worst-case effects.

8. Again, this value was purposely set to an upper bound in order to test worst-case effects.

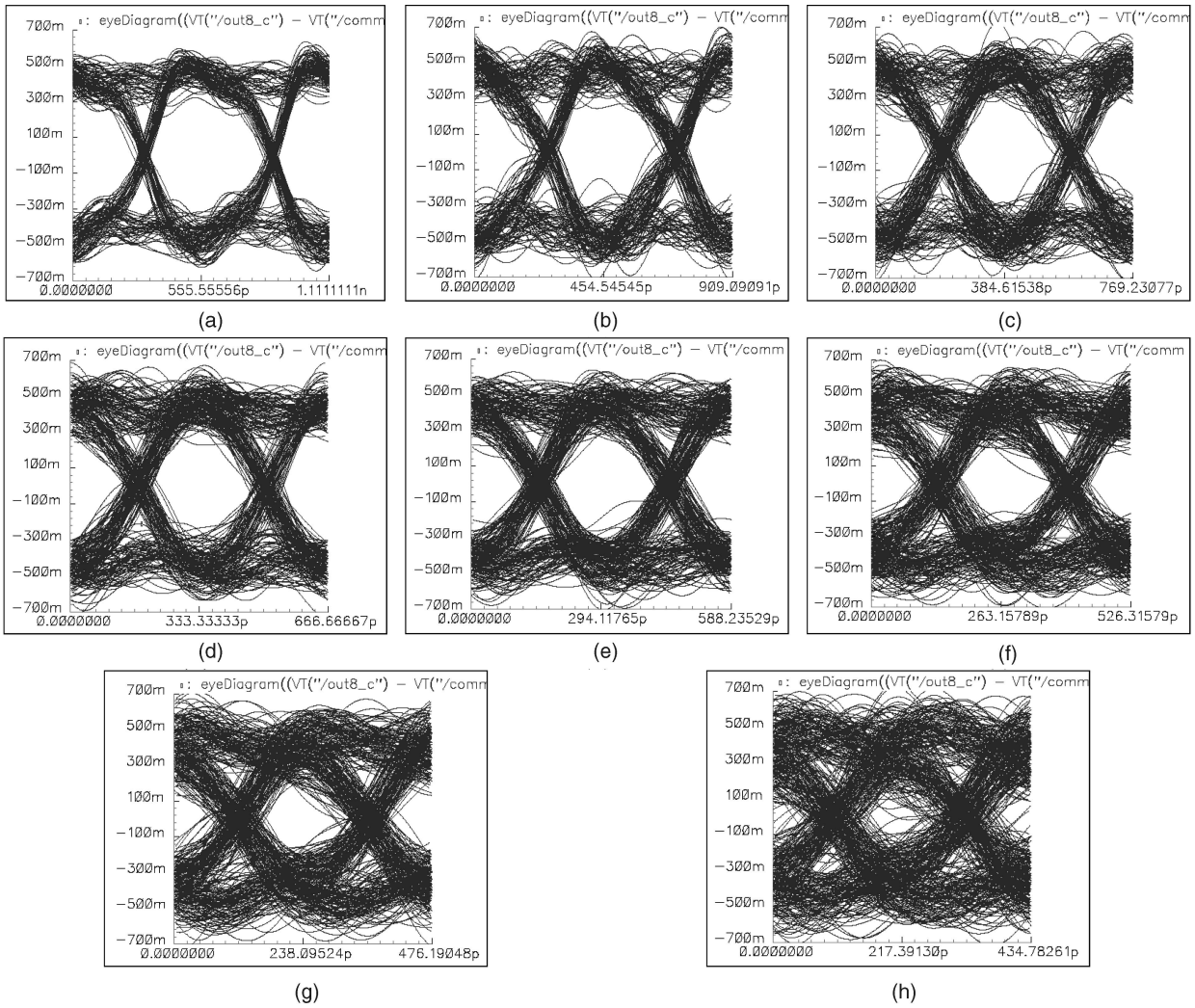

Fig. 8. Eye diagrams of one receiver circuit input as symbol transmission rate is scaled along the following scale: (a) 1.8 GHz, (b) 2.2 GHz, (c) 2.6 GHz, (d) 3.0 GHz, (e) 3.4 GHz, (f) 3.8 GHz, (g) 4.2 GHz, and (h) 4.6 GHz.

Table 6 defines three interconnect configurations that are designed to have minimum information and decoding overhead. Each of these interconnects is capable of correcting one nCm symbol containing a single bit error (high-level code is a $(n = 3, k = 2, d = 2, q = s)$-checksum and each symbol set is partitioned for distance 4). The other two interconnects and associated LHECC parameters are designed to have higher overhead and multisymbol error correcting capability. Table 7 shows an interconnect configuration that is capable of correcting up to two symbols having one bit error each (high-level code is an $(n = 4, k = 2, d = 3, q = 3)$-MDS code and each symbol set is partitioned for distance 4). Table 8 shows an interconnect configuration that is capable of correcting up to two symbols having up to two bit errors each, assuming the errors yield invalid nCm symbols or one symbol having two bit errors when the errors yield a valid nCm symbol (high-level code is an $(n = 4, k = 2, d = 3, q = 9)$-MDS code and the symbol set is partitioned for distance 6).

In Tables 6, 7, and 8, the "interconnect" column shows the number and type of MBDS channel. The "width" column shows the overall width of the interconnect in the total number of wires. The "capacity (no-ecc)" column shows how many bits of source data are carried by the interconnect without LHECC. The absolute code rate is also indicated in this column, computed as the bit capacity of the interconnect divided by the total number of wires. The "capacity (ecc)" column shows how many bits of source data are carried by the interconnect with LHECC, as well as the corresponding absolute code rate. The "rel. code rate" column shows the relative code rate for the chosen LHECC parameters, which is computed as the number of bits carried by the interconnect with LHECC divided by the number of bits carried by the interconnect without LHECC. This value represents the amount of relative information overhead required for LHECC. The overhead in bits is the difference in capacity between the interconnect without LHECC and the interconnect with LHECC.

4.5 Experimental Results

The code word error counts for each interconnect configuration with each type of noise configuration over a range of code word transmission rates are shown in Tables 9, 10, 11, 12, and 13. A zero error count for any particular simulation run means that no code word errors occurred

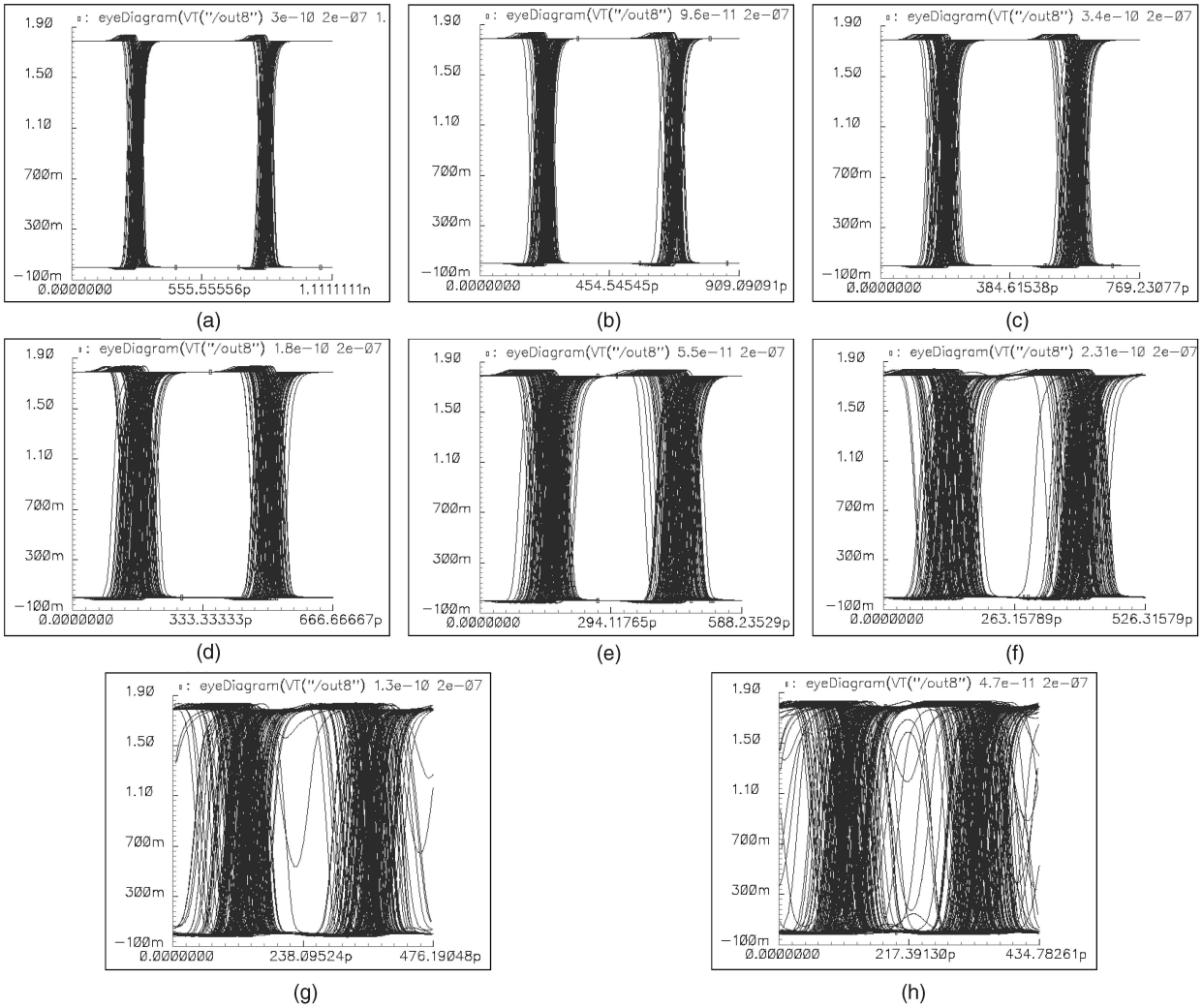

Fig. 9. Eye diagrams of one receiver output as symbol transmission rate is scaled along the following scale: (a) 1.8 GHz, (b) 2.2 GHz, (c) 2.6 GHz, (d) 3.0 GHz, (e) 3.4 GHz, (f) 3.8 GHz, (g) 4.2 GHz, and (h) 4.6 GHz.

over the 100,000 code word simulation run. This may be interpreted as a code word error rate of less than 10^{-5} for the sampled simulation time, whereas a single digit error count may be interpreted as a symbol error rate of 10^{-5}, a double-digit error count as 10^{-4}, etc.

The most important question that these results must answer is the following: *Given a predefined tolerable symbol error rate and its corresponding maximum code word transmission rates (switching frequency) for both the raw and error control-encoded interconnects, does the relative increase in code word transmission rate facilitated by the addition of error control coding yield a higher effective end-to-end interconnect throughput given the information overhead required by the code?* With traditional bit-serial communication channels

TABLE 6
Single-Symbol Correcting Interconnects

interconnect	width	capacity (no-ecc)	capacity (ecc)	rel. code rate
3x4c2	12	6/.50	6/.50	1.00
3x6c3	18	12/.67	10/.56	.83
3x8c4	24	18/.75	15/.63	.83

The LHECC configuration associated with these interconnects is capable of correcting one symbol containing one bit error.

TABLE 7
Multiple-Symbol Correcting Interconnect

interconnect	width	capacity (no-ecc)	capacity (ecc)	rel. code rate
4x4c2	16	8/.50	7/.44	.88

The LHECC configuration associated with this interconnect is capable of correcting up to two symbols containing one bit error each.

TABLE 8
Multiple-Symbol Correcting Interconnect

interconnect	width	capacity (no-ecc)	capacity (ecc)	rel. code rate
4x6c3	24	16/.67	10/.42	.63

The LHECC configuration associated with this interconnect is capable of correcting up to two symbols containing up to two bit errors each or one symbol containing up to two bit errors if the result of the errors yield a valid nCm symbol.

TABLE 9
Code Word Error Counts for Single-Symbol Correcting Interconnect Formed with Three Parallel **4c2** Channels with (a) Channel Noise Sources Only, (b) Generated Supply Noise, and (c) Additional Fringe Capacitance

rate (GHz)	# of code word errors over 100,000 code word transmissions: raw channel	channel with ecc
3.4	0	0
3.6	8	0
3.8	10	0
4.0	27	0
4.2	625	0
4.4	4176	65
4.6	11967	711
4.8	25953	3533
5.0	40311	9178
5.2	53458	17779
5.4	64391	27943
5.6	70632	36235
5.8	75781	43137

(a)

rate (GHz)	# of code word errors over 100,000 code word transmissions: raw channel	channel with ecc
3.2	0	0
3.4	4	0
3.6	8	0
3.8	87	0
4.0	532	0
4.2	3358	18
4.4	8247	83
4.6	14711	1235
4.8	29212	4646
5.0	40722	9358
5.2	55243	18919
5.4	64961	28659
5.6	71727	37380

(b)

rate (GHz)	# of code word errors over 100,000 code word transmissions: raw channel	channel with ecc
2.8	0	0
3.0	5	0
3.2	15	0
3.4	40	0
3.6	196	0
3.8	2149	0
4.0	6208	365
4.2	13660	1655
4.4	23048	4580
4.6	34247	9434
4.8	45957	16502
5.0	56429	24612
5.2	62934	30511

(c)

TABLE 10
Code Word Error Counts for Single-Symbol Correcting Interconnect Form with Three Parallel **6c3** Channels with (a) Channel Noise Sources Only, (b) Generated Supply Noise, and (c) Additional Fringe Capacitance

rate (GHz)	# of code word errors over 100,000 code word transmissions: raw channel	channel with ecc
3.0	0	0
3.2	12	0
3.4	26	0
3.6	90	0
3.8	412	0
4.0	1909	0
4.2	7617	276
4.4	15909	1255
4.6	30794	4753
4.8	48709	13453
5.0	64703	26527
5.2	74982	39311
5.4	81628	50318
5.6	85532	58432

(a)

rate (GHz)	# of code word errors over 100,000 code word transmissions: raw channel	channel with ecc
2.8	0	0
3.0	5	0
3.2	10	0
3.4	58	0
3.6	356	0
3.8	3012	0
4.0	6430	305
4.2	10892	818
4.4	19451	1981
4.6	33100	5819
4.8	51574	15711
5.0	65485	27442
5.2	75271	40111
5.4	82048	51004
5.6	86184	59734

(b)

rate (GHz)	# of code word errors over 100,000 code word transmissions: raw channel	channel with ecc
2.6	0	0
2.8	8	0
3.0	21	0
3.2	54	0
3.4	253	0
3.6	1771	0
3.8	6288	554
4.0	14645	1866
4.2	28082	6006
4.4	41101	12217
4.6	54138	21046
4.8	65920	32109

(c)

using traditional classes of error control codes, the answer to this question is invariably yes. However, MBDS channels consist of a group of wires that are tightly coupled through their termination network as well as through their associated fringe capacitance and mutual inductance. Therefore, the effects of noise affecting any single aspect of the channel may propagate throughout the entire channel. Therefore, the question that we seek to answer is whether a bit-level error correcting code would be beneficial for interconnects composed of such tightly coupled multibit channels.

Given the relatively short sampling period of our simulations, we can perform a rough approximation to answer this question for LHECC. For example, consider the single-symbol-correcting 8c4 interconnect with external supply noise (Table 11b), the raw interconnect has a maximum transmission rate of 2.6 Giga-code words/second before reaching a code word error rate of 10^{-5}. As shown in Table 6, without error control code overhead, each code word carries 18 bits, yielding an effective end-to-end throughput of 46.8 Gigabits/second. However, the equivalent error control coded interconnect has a maximum

TABLE 11
Code Word Error Counts for Single-Symbol Correcting Interconnect Formed with Three Parallel **8c4** Channels with (a) Channel Noise Sources Only, (b) Generated Supply Noise, and (c) Additional Fringe Capacitance

rate (GHz)	# of code word errors over 100,000 code word transmissions: raw channel	channel with ecc
2.8	0	0
3.0	28	0
3.2	52	0
3.4	196	0
3.6	641	0
3.8	1706	0
4.0	5318	132
4.2	14305	852
4.4	28462	3692
4.6	46356	11125
4.8	64278	25070
5.0	77842	41802
5.2	85862	55414
5.4	90650	66632

(a)

rate (GHz)	# of code word errors over 100,000 code word transmissions: raw channel	channel with ecc
2.6	0	0
2.8	10	0
3.0	20	0
3.2	83	0
3.4	952	0
3.6	2941	0
3.8	6475	0
4.0	9888	397
4.2	17194	1628
4.4	30778	4904
4.6	48757	13168
4.8	66652	27221
5.0	78475	42256
5.2	85990	56129
5.4	90848	67385

(b)

rate (GHz)	# of code word errors over 100,000 code word transmissions: raw channel	channel with ecc
2.2	0	0
2.4	8	0
2.6	10	0
2.8	34	0
3.0	102	0
3.2	342	0
3.4	865	0
3.6	2380	0
3.8	7303	350
4.0	18848	2204
4.2	33597	7060
4.4	50355	16217
4.6	65703	29596
4.8	77086	44015

(c)

TABLE 12
Code Word Error Counts for Double-Symbol Correcting Interconnect Formed with Four Parallel **4c2** Channels with (a) Channel Noise Sources Only, (b) Generated Supply Noise, and (c) Additional Fringe Capacitance

rate (GHz)	# of code word errors over 100,000 code word transmissions: raw channel	channel with ecc
3.4	0	0
3.6	6	0
3.8	10	0
4.0	33	0
4.2	643	0
4.4	3469	0
4.6	10921	192
4.8	23785	1048
5.0	38819	3122

(a)

rate (GHz)	# of code word errors over 100,000 code word transmissions: raw channel	channel with ecc
3.0	0	0
3.2	2	0
3.4	4	0
3.6	8	0
3.8	18	0
4.0	215	0
4.2	3234	0
4.4	7362	0
4.6	13714	221
4.8	26205	1142
5.0	39864	3322

(b)

rate (GHz)	# of code word errors over 100,000 code word transmissions: raw channel	channel with ecc
2.6	0	0
2.8	2	0
3.0	5	0
3.2	10	0
3.4	25	0
3.6	112	0
3.8	1995	0
4.0	6784	313
4.2	14821	1215
4.4	23435	2671

(c)

transmission rate of 3.8 Giga-code words/second before reaching a code word error rate of 10^{-5}. With error control code overhead, each code word carries 15 bits, yielding an effective end-to-end throughput of 57 Gigabits/second, for an effective speedup of 1.22. Performing the same analysis for a symbol error rate of 10^{-3} yields a convergence in effective throughput. As with traditional communication channels, the effective increase in throughput decreases with higher tolerable error rates. However, the example error rates given in this section are unreasonably high, suggesting a higher increase in interconnect throughput (speedup) for lower tolerable error rates. The remainder of the results show that all interconnect configurations over all noise models exhibit a speedup in end-to-end throughput for a fixed code word error rate of 10^{-5} when LHECC is utilized.

The results indicate a uniform speedup in end-to-end throughput for the error-control-encoded interconnect over the corresponding raw, non-error-control-encoded interconnect for all of the single-symbol-correcting codes, and slightly better speedup for the double-symbol-correcting codes. However, when different interconnect configurations are compared, overall throughput efficiency (throughput contribution per wire) increases with overall code density, giving an advantage to wider interconnects. Therefore, each

TABLE 13
Code Word Error Counts for Double-Symbol Correcting Interconnect Formed with Four Parallel **6c3** Channels with (a) Channel Noise Sources Only, (b) Generated Supply Noise, and (c) Additional Fringe Capacitance

rate (GHz)	# of code word errors over 100,000 code word transmissions: raw channel	channel with ecc
2.6	0	0
2.8	1	0
3.0	2	0
3.2	9	0
3.4	23	0
3.6	92	0
3.8	263	0
4.0	2597	0
4.2	9940	0
4.4	20503	0
4.6	38608	481
4.8	59413	2315

(a)

rate (GHz)	# of code word errors over 100,000 code word transmissions: raw channel	channel with ecc
2.4	0	0
2.6	2	0
2.8	3	0
3.0	7	0
3.2	18	0
3.4	62	0
3.6	439	0
3.8	2630	0
4.0	8324	0
4.2	13452	0
4.4	24512	200
4.6	41876	829
4.8	72459	3012

(b)

rate (GHz)	# of code word errors over 100,000 code word transmissions: raw channel	channel with ecc
2.2	0	0
2.4	5	0
2.6	8	0
2.8	16	0
3.0	24	0
3.2	56	0
3.4	194	0
3.6	1415	0
3.8	5286	0
4.0	15439	0
4.2	29687	400
4.4	46078	1226
4.6	62071	3449

(c)

of these interconnect configurations represents a fundamental trade-off in decoder logic complexity and I/O pins versus end-to-end throughput.[9] This trade-off represents a design space for an interconnect designer.

5 Conclusion

This paper describes Lightweight Hierarchical Error Control Codes, a technique for applying error control to high-performance system-level interconnect with low requirements for information and logic overhead. This paper illustrates how LHECC is effective at achieving low overhead error control for system-level interconnects built with Multi-Bit Differential Signaling (MBDS) channels. The sample implementations show that LHECC encoders and decoders are viable for integration into system-level interfaces due to their low requirement for chip area and capability of operating at speeds that match or exceed the operating speeds of current and next-generation system-level interconnects. Finally, interconnect simulation verifies that error control coding over MBDS channel technology exhibits end-to-end throughput improvements that are consistent with traditional error correction encoding for traditional communication channels.

References

[1] D.M. Chiarulli, J.D. Bakos, J.R. Martin, and S.P. Levitan, "Area, Power, and Pin Efficient Bus Transceiver Using Multi-Bit-Differential Signaling," *Proc. IEEE Int'l Symp. Circuits and Systems (ISCAS '05)*, May 2005.

[2] *IEEE Standard for Low-Voltage Differential Signaling (LVDS) for Scalable Coherent Interface (SCI)*, 1596.3 SCI-LVDS standard, IEEE Std. 1596.3-1996, 1996.

[3] A. Narasimhan, M. Kasotiya, and R. Sridhar, "A Low-Swing Differential Signalling Scheme for On-Chip Global Interconnects," *Proc. 18th Int'l Conf. VLSI Design*, pp. 634-639, Jan. 2005.

[4] S. Hirsch and H.-J. Pfleiderer, "CMOS Receiver Circuits for High-Speed Data Transmission According to LVDS-Standard," *Proc. SPIE*, vol. 5117, 2003.

[5] M. Kahrs, S.P. Levitan, D.M. Chiarulli, T.P. Kurzweg, J.A. Martínez, J. Boles, A.J. Davare, E. Jackson, C. Windish, F. Kiamilev, A. Bhaduri, M. Taufik, X. Wang, A. Morris, J. Kruchowski, and B.K. Gilbert, "System-Level Modeling and Simulation of the 10G Optoelectronic Interconnect," *IEEE J. Selected Topics in Quantum Electronics*, vol. 21, no. 12, pp. 3244-3256, Dec. 2003.

[6] Cadence Design Systems, http://www.cadence.com, 2007.

[7] M. Saint-Laurent and M. Swaminathan, "Impact of Power-Supply Noise on Timing in High-Frequency Microprocessors," *IEEE Trans. Advanced Packaging*, vol. 27, no. 1, pp. 135-144, Feb. 2004.

[8] Private correspondence with Martin Saint-Laurent, Intel Corp.

[9] *Xilinx RocketIO Transceiver User Guide*, http://direct.xilinx.com/bvdocs/userguides/ug024.pdf, Dec. 2004.

[10] Digital Visual Interface, Revision 1.0, Digital Display Working Group (DDWG), http://www.ddwg.org/lib/dvi_10.pdf, Apr. 1999.

[11] C. Sauer, M. Gries, J.I. Gomez, S. Weber, and K. Keutzer, "Developing a Flexible Interface for RapidIO, Hypertransport, and PCI-Express," *Proc. Int'l Conf. Parallel Computing in Electrical Eng.*, pp. 129-134, Sept. 2004.

[12] R. Ho, K.W. Mai, and M.A. Horowitz, "The Future of Wires," *Proc. IEEE*, vol. 89, no. 4, pp. 490-504, Apr. 2001.

[13] D.M. Chiarulli, J.D. Bakos, J.R. Martin, and S.P. Levitan, "Area, Power, and Pin Efficient Bus Structures Using Multi-Bit-Differential Signaling," *Proc. SPIE Europe Int'l Symp.: Microtechnologies for the New Millennium*, May 2005.

[14] S.P. Levitan, D.M. Chiarulli, S. Dickerson, J. Bakos, and J. Martin, "Power Efficient Communication Using Multi-Bit-Differential Signaling," *Proc. 16th Ann. IEEE-LEOS Workshop Interconnections within High-Speed Digital Systems*, May 2005.

9. Channel capacity, in terms of number of bits transferred over the interconnect in one clock cycle, is also a parameter upon which a designer may place high importance. For example, router chips communicating on a wormhole-switched network may need to design router buffers and internal crossbar switches according to the atomic transfer unit of the network's router-to-router interconnects. The designer may also wish to bundle control signals along with the data signals within the interconnect.

Jason D. Bakos received the BS degree in computer science from Youngstown State University in 1999 and the PhD degree in computer science from the University of Pittsburgh in 2005. He worked as a research and teaching assistant at the University of Pittsburgh from 1999 to 2005 and currently serves as an assistant professor in the Department of Computer Science and Engineering at the University of South Carolina. He is a reviewer for the International Symposium of Circuits and Systems and for John Wiley & Sons. He has published articles in publications spanning a broad range of diverse disciplines, including circuits and systems, optoelectronics, and reconfigurable computing. He was the recipient of DAC/ISSCC/IEEE/ACM design contest awards at the Design Automation Conference in 2002 and 2004. He is a member of the IEEE, ACM, and IEEE Computer Society.

Donald M. Chiarulli received the MS degree in computer science from the Virginia Polytechnic Institute in 1979 and the PhD degree, also in computer science, from Louisiana State University in 1986. He is currently a professor of computer science at the University of Pittsburgh. His research interests are in computer architecture with a specific focus on technology and architecture for interconnects. Contributions from his group have included the demonstration of the first all-optical address decoder, several designs for time/space multiplexed data bus architectures, and an all-optical distributed bus arbitration algorithm. Later contributions include the Partitioned Optical Passive Star (POPS) architecture for multiprocessor interconnection networks. Other contributions from his group have included a prediction method for multiprocessor memory access patterns and contributions to the development of a set of tools for optoelectronic CAD. Both of these efforts have earned best paper awards at the International Conference on Neural Networks (ICNN) and the Design Automation Conference (DAC), respectively. He is a member of the IEEE, OSA, and SPIE.

Steven P. Levitan received the BS degree from Case Western Reserve University in 1972. He received the MS degree in 1979 and the PhD degree in 1984, both in computer science, from the University of Massachusetts, Amherst. From 1972 to 1977, he worked for Xylogic Systems designing hardware for computerized text processing systems. He was an assistant professor from 1984 to 1986 in the Electrical and Computer Engineering Department at the University of Massachusetts. In 1987, he joined the Electrical and Computer Engineering faculty at the University of Pittsburgh, where he is the John A. Jurenko Professor of Computer Engineering in the Department of Electrical and Computer Engineering and holds a joint appointment in the Department of Computer Science. He is past chair of the ACM Special Interest Group on Design Automation (SIGDA). He is a member of the ACM/IEEE Design Automation Conference Executive Committee. He is a senior member of the IEEE and a member of SPIE and OSA.