LIGHTWEIGHT HIERARCHICAL ERROR CONTROL CODES FOR MULTI-BIT DIFFERENTIAL CHANNELS

by

Jason D. Bakos

BS, Youngstown State University, 1999

Submitted to the Graduate Faculty of

Arts and Sciences in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2005

UNIVERSITY OF PITTSBURGH

FACULTY OF ARTS AND SCIENCES

This dissertation was presented

by

Jason D. Bakos

It was defended on

June 28, 2005

and approved by

Donald M. Chiarulli

Bruce R. Childers

Steven P. Levitan

Patchrawat Uthaisombut

Donald M. Chiarulli
Dissertation Director

LIGHTWEIGHT HIERARCHICAL ERROR CONTROL CODES
FOR MULTI-BIT DIFFERENTIAL CHANNELS

Jason D. Bakos, Ph.D.
Advisor: Donald M. Chiarulli

University of Pittsburgh, 2005

This dissertation describes a new class of non-linear block codes called "Lightweight Hierarchical Error Control Codes (LHECC)." LHECC is designed to operate over system-level interconnects such as network-on-chip, inter-chip, and backplane interconnects. LHECC provides these interconnects with powerful error correction capability and thus effectively increases signal integrity and noise immunity. As a result, these interconnects may carry data with lower signal power and/or higher transmission rates. LHECC is designed such that support for it may be tightly integrated into high-speed, low-latency system-level I/O interfaces. Encoding and decoding may be performed at system core speeds with low chip area requirements.

LHECC is optimized for a new type of high-performance system-level interconnect technology called Multi-Bit Differential Signaling (MBDS). MBDS channels require the use of 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 1-bits. These symbol sets have properties such as inherent error detection capability and unused symbol space. These properties are used to give MBDS-based system-level interconnects an arbitrary error correction capability with low or zero information overhead. This is achieved by hierarchical encoding, where a portion of source data is encoded into a "high-level" block code while the remainder of the data is encoded

into a “low-level” code by choosing particular nCm symbols from symbol subsets specified by the high-level encoding.

This dissertation presents the following. First, it provides a theoretical study of LHECC and illustrates its effectiveness at achieving low-overhead error control for system-level interconnects. Second, it provides example implementations of efficient LHECC encoder and decoder architectures that are capable of operating at speeds necessary for high-performance system-level channels. Third, it describes an experimental technique to verify the effectiveness of these codes, where interconnect error behavior is captured using channel and noise models over a range of transmission rates and noise characteristics. Results obtained through simulation of these models characterize the effectiveness of LHECC. Using this method, system-level interconnects that utilize this new encoding technique are shown to have significantly higher noise immunity than those without.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

This dissertation represents the conclusion of a six-year journey of intense study, academic growth, and preparation for a career of scientific research and other scholarly endeavors. I would like to thank my advisors, Dr. Donald M. Chiarulli and Dr. Steven P. Levitan, for their tireless years of close guidance. I also would like to thank the other members of my Ph.D. committee, Dr. Bruce Childers and Dr. Patchrawat Uthaisombut, who were instrumental in helping me focus in on the goals of this work and were crucial to its success. Lastly, I thank my wife, Lumi, for her continuous loving support throughout my graduate career. Lumi is the true source of my inspiration.

1. INTRODUCTION

This dissertation describes a new class of error control codes (ECC) called *Lightweight Hierarchical Error Control Codes (LHECC)*. LHECC is designed such that it may be efficiently applied over *system-level interconnects (SLI)* built with *Multi-Bit Differential Signaling (MBDS)* technology. LHECC endows these interconnects with powerful error correction capability while requiring low information overhead. This enables these interconnects to achieve high data throughput while requiring low encoding and decoding complexity. LHECC may be efficiently applied to system-level I/O interfaces and it is capable of correcting the types of errors that affect system-level interconnects. The benefits of LHECC support is verified and measured through simulation of interconnect and noise models.

Lightweight Hierarchical Error Control Codes are based on non-linear block codes and rely 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. This allows effective data throughput to closely approach or match the theoretical maximum transmission rate of the channel.

Lightweight Hierarchical Error Control Codes require low encoding/decoding logic complexity through their use of small block lengths and hierarchical encoding and decoding. In traditional block codes, longer block lengths are used to reduce relative information overhead at the cost of

higher decoding complexity. LHECC has lower information overhead relative to traditional block codes even when using equivalent block lengths. 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 complexity while simultaneously exhibiting powerful error correction capability and low end-to-end latency. LHECC encoding and decoding may be performed at system core speeds and with low chip real estate requirements. These encoders and decoders require only a few hundred standard logic gates and latches. The result is increased signal integrity with the addition of only digital components to the channel interfaces. Furthermore, when building an interconnect with LHECC support, the designer may set several parameters that allow for trade-offs among interconnect width, information overhead, error correction power, and encoder/decoder complexity.

Lightweight Hierarchical Error Control Codes are designed to operate over Multi-Bit Differential Signaling (MBDS), which is a pin- and power-efficient extension to traditional pair-wise differential signaling. MBDS is a next-generation technology for building high-performance channels for system-level interconnects, such as network-on-chip, inter-chip, backplane, and high-speed peripheral interconnect. 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 (nCm)*" encoding. The LHECC technique extends MBDS by placing rules on how this encoding is performed over parallel MBDS channels. These encoding rules allow the receiver to correct transmission errors that occur as a result of noise affecting the channel.

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 exactly half the bits are binary-1 values and the other half are binary-0 values. This channel code offers built-in information redundancy through unmapped symbols and inherent error detection capability. These properties are used to amortize most if not all of the information overhead required to achieve channel error correction capability. These properties are also used to reduce encoding and decoding complexity relative to a traditional block code of the same block length. When LHECC is applied over several parallel nCm-encoded MBDS channels the result is a binary ECC code space.

In order to measure the effectiveness of the LHECC technique, interconnect and noise models are used to characterize the types of errors that are likely to occur within MBDS-based interconnect. The interconnect models are formed by several parallel channel models, where the number of channel models is dictated by the configuration of the LHECC. Each channel model consists of driver and receiver circuit models connected by chip package and PCB trace transmission line models. The interconnect models themselves contain error sources originating from transistor noise, signal crosstalk, jitter, and frequency-domain signal attenuation. Other error sources, such as power supply noise and fringe capacitance, are also modeled and added to the interconnect model. These models are simulated with various LHECC configurations and channel transmission rates in order to determine the relative number of correctable error occurrences. These results measure the decrease in error ratio for interconnects with LHECC

support relative to interconnects without. These experiments determine that the benefits of LHECC, in terms of added signal integrity, outweigh its costs in implementation.

This dissertation begins with the motivation and background of the LHECC technique. Next, it describes the LHECC technique and provides example implementations of LHECC encoders and decoders. After this, it presents modeling and simulation techniques that are used to characterize the level of noise immunity gained from applying LHECC over MBDS channels. Finally, it presents and discusses the results obtained from these verification techniques.

2. MOTIVATION

This work is motivated by the fact that traditional error control codes are not well-suited for system-level interconnect (SLI). This is because traditional error control codes have logic, latency, and information overhead requirements that make these codes unattractive for system-level applications. There are several reasons for this. First, system-level interfaces cannot accommodate the large amount of support logic and memory required for these codes. Second, the high overhead requirement of these codes consumes much of the available channel capacity. To illustrate these points, this chapter describes traditional error control codes and the types of channels over which they are used. Next, it provides an overview of the properties and challenges of system-level interconnect.

2.1. Traditional Error Control Codes

The work presented in this dissertation builds on concepts from error control code theory. Error control codes (ECC) are an information-theoretic technique for increasing signal integrity in communication channels and storage devices. Their goal is to provide a mechanism for detecting and correcting infrequent signal errors that are introduced within these channels as a result of noise. In many cases, the class of ECC that is selected for a particular channel is driven by characteristics of the channel such as channel length, signaling frequency/transmission rate, and signal-to-noise ratio. These characteristics dictate the types and frequency of errors that are expected to occur, which in turn is a function of the noise sources that affect the channel.

However, the benefits of error control encoding come at a cost. ECC requires that data be *encoded* prior to transmission. Encoding involves computing redundant information, called *parity*, as a function of source data. This parity information is used for error detection and correction, and is computed and appended to the source data prior to transmission. Parity consumes channel capacity but does not carry any actual source data. In general, the code's error correction power as well as its encoding and decoding complexity increases with the amount of parity (and thus the amount of *overhead*) that is added to the source data.

Upon reception, data must be *decoded* in order for the receiver to detect and possibly correct errors. ECC codes that are in widespread use, such as Reed-Solomon Codes [39] and Turbo Codes [67], are well-suited for use over long-haul, low-speed, low-signal-power, and/or noisy channels. This is due to the types of errors these codes are designed to correct as well as the amount of encoding and decoding complexity that can be afforded in these types of channels. Types of channels where these codes are applied include deep-space signaling, cellular networks, digital television transmission, and magnetic and optical storage mediums [39]. Traditional classes of ECC require a significant amount of information and/or computational overhead. As a result, encoding and decoding is typically performed with microcontroller software or dedicated devices. Commercially available high-speed error control devices and IP cores are limited to maximum throughputs on the order of megabytes-per-second and latencies on the order of microseconds [85, 93, 94]. Two classes of ECC codes that are in wide-spread use are convolutional codes and block codes.

Convolutional codes, such as those used as a component of Turbo Codes, are powerful codes that offer a high level of error correction capability [84]. These codes add parity bits to each symbol of source data prior to transmission. The width of each symbol is typically comprised of one to four bits. The encoding of each symbol is a function of the symbol itself as well as previously transmitted symbols (depending on the code's *constraint length*). The width of each symbol, the number of parity bits per symbol, and constraint length are parameters of the code. These codes are used in channels with low signal-to-noise ratio and thus where signal reliability is more important than high data throughput. Examples of such channels are those used in cellular networks and deep-space signaling. These channels have low signal power and are highly susceptible to ambient RF noise making signal errors relatively common. However, while convolutional encoders and decoders are relatively simple to implement, convolutional codes require a large amount of relative information overhead. For example, the convolutional code used to re-encode the block ECC-encoded data of the Mars Pathfinder, Mars Exploration Rover, and the Cassini probe to Saturn require transmission of over six bits in order to encode a single bit [86, 92]. The effect of this overhead is that the actual data throughput is significantly lower than the base transmission rate of the channel.

Block codes, such as Reed-Solomon codes, are used for channels and storage devices that are higher power, less noisy, and/or require higher throughput [84]. These codes operate at a symbol granularity, where each symbol typically consists of 8-bits (1 byte). These codes require that several parity symbols be computed as a function of each segment of source data. These segment widths typically range from 10 to 200 symbols. The length of each segment and number of parity symbols are parameters of the code. Each segment of source symbols is

combined with its associated parity symbols to form a *block*. Each block is encoded independently from all other blocks. Block codes are used in digital television transmission and magnetic/optical storage devices (where media damage causes storage errors). Block codes also require that a high amount of information overhead be added to the source data, as several parity symbols are required for each block. However, high throughput is achieved in these channels by keeping the relative amount of information overhead low by selecting codes with a large block sizes. For example, HDTV transmission standards add approximately 20 bytes of parity to every 200 bytes of source data, yielding block sizes of 220 bytes and thus requiring only 10% information overhead. This allows the data throughput to closely approach the base transmission rate of the channel. However, decoding these codes requires that the entire block be received before error detection and error correction can take place. This results in high transmission latency and requires that the decoder contain at least enough memory to buffer one complete block, implying high memory and latency requirements. Also, decoding codes with large block sizes requires substantial logic complexity, implying high chip real estate and power requirements. For example, the following experimental decoders for Reed-Solomon codes are described in the literature as being area efficient: (149, 138) decoder requiring 35,000 logic gates [95], scalable decoder requiring 1 mm^2 of chip area and 49,000 transistors [96], decoder requiring 2.90 mm x 2.88 mm of chip area and 26,000 gates [97], (255, 239) decoder requiring 115,500 logic gates [98], scalable decoder requiring 502,896 to 1,007,356 transistors depending on error correction power [99], (255,239) decoder requiring 32,900 logic gates and 2.03 mm^2 of chip area [100], and a (207,187) decoder requiring 43,000 logic gates and 7000 clock cycles to decode one block [101]. For these types of channels, the requirements for memory and decoding complexity are acceptable because effective data transmission rates are fundamentally limited by

the channel to the high megabit-per-second or low gigabit-per second range. This allows the use of dedicated encoder and decoder devices that require multiple core clock cycles to encoder or decode each block.

Traditional types of error control codes are not appropriate for SLI channels because high data throughput and low latency are crucial for these channels. The reasons for this requirement are described in the next section. In addition, when applying error control codes to SLI channels, the encoders and decoders must physically fit within the real estate reserved for system-level interfaces (i.e. a network-on-chip switching element or off-chip I/O pad frame). Furthermore, decoding should be performed and pass data through within each core clock cycle (the finest granularity of time within a computer system). This is to allow individual signal transmission frequencies to match core logic speeds that reach into the multiple gigahertz range. As a result, total aggregate data throughput for the interconnect may reach into the hundreds of gigabits per second.

2.2. System-Level Interconnect

System-level interconnects connect major components of a digital system. Processor-to-memory interconnect is the most widely-known example of this type of interconnect. The channels used for these types of interconnects are characterized by their extremely high transmission rate (and thus signal frequency) and short channel length as compared to other types of communication channels.

Structurally, SLI is based on *short-haul signaling*, meaning that components sharing the channel are physically separated on a scale of tens or hundreds of millimeters. SLI channels are designed

to carry binary data and control signals and are usually formed by multiple signal conductors in parallel. SLI is an integral component to a computer system, and thus its performance constitutes a major factor in overall system performance. Because of this, designers of SLI technologies seek to achieve the highest possible aggregate data throughput and lowest possible end-to-end latency offered by the available fabrication technology. Many types of SLI, particularly computer memory busses, achieve high throughput through the use of wide channels (over 100 wires wide). System busses of these widths may not be feasible in future systems. For example, future single-chip microprocessors (CMP) may contain 10-100 processor cores on a single chip, where each processor core must have independent system interfaces that must share a single chip's I/O resources. Such a situation would call for chips with tens of thousands of signal I/O pins. In response to such problems, industry trends for next-generation systems call for high-speed serialized and multiplexed SLI channels for more efficient use of chip resources such a power and I/O pins.

Other examples of SLI include global on-chip switching/routing fabrics for network-on-chip (NoC) systems for chip multiprocessors and other system-on-a-chip designs [84]. A more classical example of SLI is inter-chip interconnect for both multi-chip modules (MCM) and printed circuit boards (PCBs) that are used for processor-to-memory interconnects. System-level interconnect techniques are also used for backplane devices for high-performance parallel computers and network switches. High-speed peripheral interconnects may also be considered system-level interconnects, particularly in applications such as high-capacity and high-speed display, storage, and sensor devices.

SLI, being comprised of high-speed and short-haul channels, differs from traditional low-speed, long-haul, bit-serial channels in the sources of noise to which it is susceptible. Rather than being independent components, SLI I/O interfaces are tightly coupled to highly integrated digital chips. As such, they are affected by the same types of noise that face large digital systems. Examples of such noise sources are timing jitter, power supply noise, and parasitic capacitive and inductive effects. Jitter is dynamic data timing inconsistency that is common in high-speed synchronous channels. Localized supply noise on chips is introduced into SLI transmitters and receivers by switching noise created by digital logic located elsewhere in the chip and acts to reduce signal margins within the transmitters and receivers. Capacitive and inductive effects are caused by both parasitic and transmission line effects created by the proximity and relative sizes of features that exist within the physical circuit layout, chip packaging, and physical conductors of the SLI channel.

As with most communication channels, SLI channel interfaces often include various analog circuit design techniques within its transmitters and receivers to maximize signal integrity, noise immunity, and signal-to-noise ratio. However, many of these techniques rely on analog circuit techniques such as transmitter pre-emphasis, receiver equalization, and signal modulation. Design of these circuits in a VLSI system is complex and requires precise tuning, careful manual layout design, and substrate isolation. The goal of the work presented in this dissertation is to improve signal integrity by utilizing data encoding techniques at the information-layer (link-layer) of SLI channels. This is a relatively new concept for SLI design and requires only the addition of digital components to the channel interfaces. The secondary goal of the work presented in this dissertation is to design SLI channels and associated support logic that are

capable of matching the clock speed and local signal transmission rate of on-chip core logic. For example, chip-to-chip channels should operate at the same signal frequency as their corresponding on-chip logic. This goal is consistent with the desire to close the growing gap between local on-chip versus system-level transmission rates that threaten to be a bottleneck in next generation systems.

Lightweight Hierarchical Error Control Codes (LHECC) are designed to operate over a newly developed high-performance system-level interconnect (SLI) technology. It also serves as a demonstration that error control encoding over SLI is both feasible and that it yields enough additional signal integrity to justify the costs of its application.

2.3. Challenges for Designing High-Performance SLI Channels

High-performance SLI signaling technology is an increasingly important research area in computer system design. This is primarily because advancements in semiconductor fabrication technology are creating a divergence between the per-wire data transfer speeds of on-chip data signals versus corresponding off-chip I/O signals. Figure 2.1 illustrates this trend by plotting the growth of core logic speeds and corresponding system bus speeds over successive generations of Intel's processor architectures. As shown in the plot, on-chip data speeds follow a roughly exponential curve (as Moore's Law predicts), while corresponding off-chip system interface speeds follow a more gradual curve. New techniques for building high-performance chip-to-chip interconnects are of increasing importance to prevent this problem from becoming a critical bottleneck in future system design.

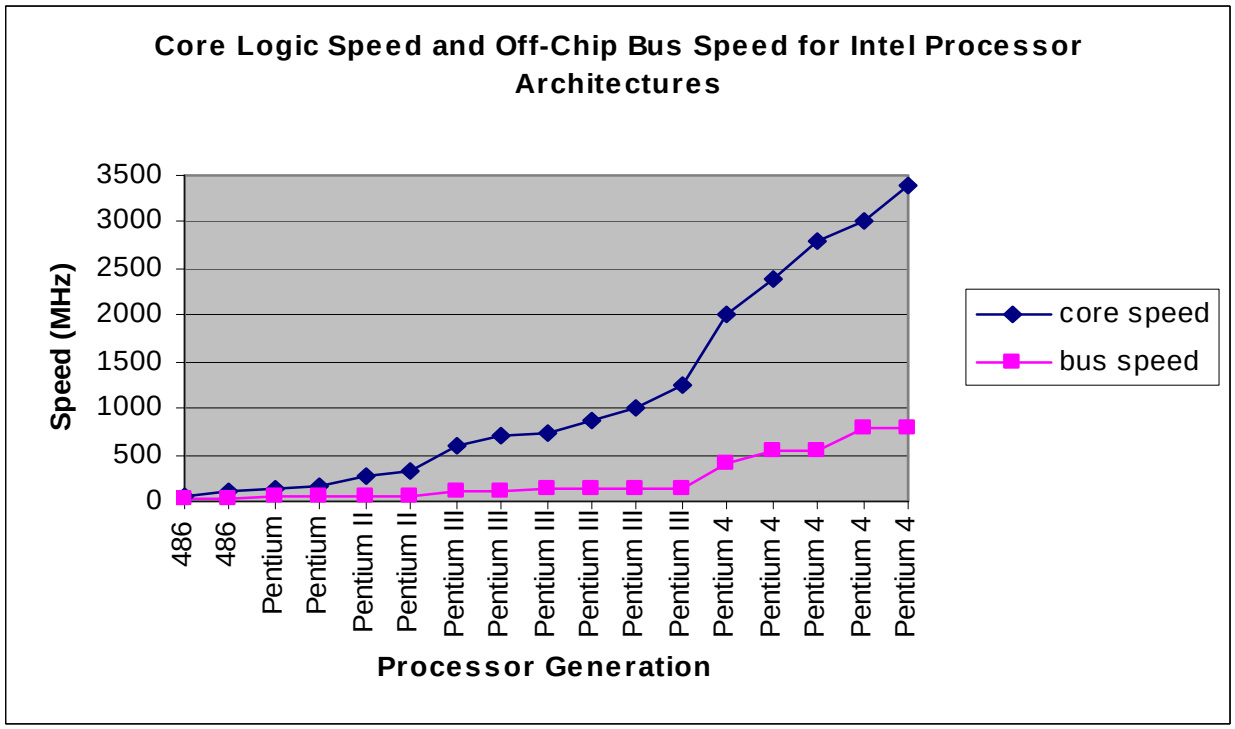


Figure 2.1: Trends for Intel processor architectures: core logic and system interface speeds (source: compiled information gathered from Intel Corporation).

There are several challenges in developing high-performance SLI technologies. Many of these challenges stem from a general motivation to minimize the chip resources reserved for signaling, such as chip real estate, power, and I/O pins. One way to achieve this is for chip designers to multiplex I/O data streams through high-speed serialized channels. There are several advantages in this technique. First, while real estate and power are important chip resources, I/O pads are often the most constrained resource. In traditional pad frames, I/O pins become more of a constraint as chip sizes grow because the perimeter of a chip grows linearly as total chip area grows quadratically. Even the size and density of flip-chip area pad arrays is limited by minimum width and spacing rules for breakout traces. To further complicate matters, as chip integration levels grow, designers are motivated to reserve a larger portion of I/O pins for power and ground connections due to high power requirements and minimization of power supply

noise. In addition, SLI driver circuits must be physically large and thus consume a large amount of chip real estate. This is because SLI drivers must drive a large capacitive and resistive load as well as compensate for an electrical phenomenon called the skin effect. The skin effect causes the effective impedance of an electrical conductor to increase with signal frequency. This creates an inverse relationship between signal frequency and signal amplitude (signal swing). High-frequency drivers must support high amplification in order to be capable of sourcing enough electrical current to compensate. The amount of current that can be drawn through driver transistors is a function of the size of the transistor. These transistors become proportionally larger in area as compared to minimum chip features over successive generations of fabrication technology. This makes them consume a large amount of power and creates another resource constraint for high-amplification drivers.

Another challenge for high-performance electrical signaling is the difficulty in maintaining signal integrity for high-frequency signals. This challenge is caused by the physical properties of the electrical connections that make up channels. Signal wires that are equal or longer in length than the associated signal wavelength are modeled as *transmission lines* [78]. In general, the effects of transmission lines make high-frequency signaling difficult. For example, a cross-sectional view of a chip-to-chip channel is shown in Figure 2.2. For this channel, the signaling path through the chip package and printed circuit board trace are modeled as a transmission line.

In general, the integrity of signals sent through transmission lines degrades as signal frequency increases. This is mostly due to the low-pass frequency response created by transmission lines. This low-pass frequency response acts to filter out or attenuate high-frequency signals and signal

components. In addition, reliable reception of high-frequency signals is affected by ambient RF noise, crosstalk between adjacent and near-by conductors, signal reflections, dynamic timing inconsistency in synchronous channels (jitter), and PCB resistive and dielectric loss. These problems become increasingly serious when signaling channels are heavily loaded by multiple drops, run adjacent to parallel channels, or distributed over relatively long distances.

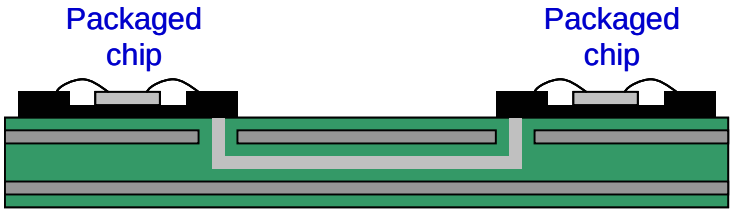


Figure 2.2: Cross-sectional view of transmission line formed between two chips mounted on a printed circuit board.

3. BACKGROUND: MULTI-BIT DIFFERENTIAL SIGNALING

Lightweight Hierarchical Error Control Codes are designed to operate over channels built with Multi-Bit Differential Signaling (MBDS) technology. This chapter describes this underlying MBDS channel technology as well as several of its properties that are used to support LHECC.

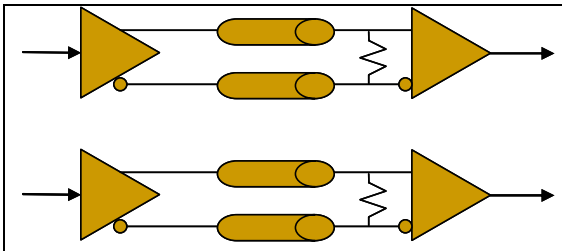
3.1. Differential Signaling

Pair-wise (single-bit) differential signaling is a well known and widely used technique for building high-performance channels for system-level interconnect (SLI). For example, Low Voltage Differential Signaling (LVDS), as defined by the ANSI/TIA/EIA-644-A standard, specifies a set of industry standard signaling specifications for differential channels. LVDS is currently used in such high-speed signaling technologies such as Xilinx RocketIO, Infiniband, Firewire (IEEE 1394), SCSI LVD, Flat Panel Display (FPD) Link, LVDS Display Interface (LDI), and OpenLDI standards [77].

An LVDS channel is formed by a two wire electrical connection between a driver and receiver. An LVDS driver is a current-mode device that transmits a binary value across the channel by steering current through one of the two wires and returning the current through the other. Near the receiver, a matched termination resistor is connected across the wires to minimize signal reflection. A positive or negative voltage will exist across this resistor depending on which direction current is flowing. A differential receiver is connected to both wires and converts and amplifies (conditions) the resulting positive or negative voltage to CMOS levels that are suitable

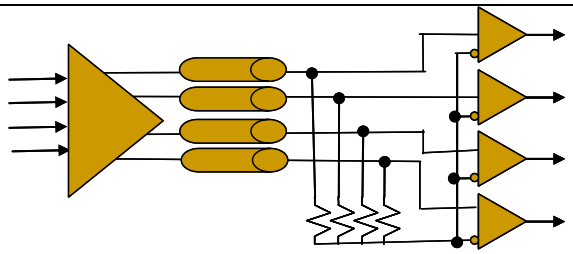
for on-chip transmission and use in digital circuits. In effect, LVDS employs a channel code where binary data is converted to *unary* data, creating a physical-layer "symbol set" of {01, 10}. The capacity of such a channel is a single bit, meaning that two I/O pins and two wires are required for each bit of capacity within the interconnect. In this case, the absolute code rate, as defined by the number of data bits divided by the number of physical connections, is 50%.

LVDS channels have many advantages over other signaling techniques. First, LVDS does not require a shared global signal reference or shared power supply. This makes LVDS channels immune to-noise entering transmitters and receivers through a global reference and allows the driver and receiver to have a large ground potential difference (GDP). Also, an LVDS driver injects relatively little "switching noise" into the channel or its own power supply because it consumes a constant amount of current regardless of whether it is transmitting a logic-0 or logic-1. Ambient RF noise affecting the physical connections manifests itself as "common-mode noise" and generally affects both physical connections equally, assuming the wires forming the connection are routed in close proximity. This type of noise is canceled out at the receiver because the receiver only senses the voltage *difference* between the two physical connections in the channel. LVDS channels also exhibit favorable coupled electromagnetic (EM) behavior between the two wires. For these reasons, LVDS enjoys a high level of noise immunity and allows LVDS channels to operate using small signal voltage swing. This acts to reduce the detrimental effect of the low-pass frequency response of the transmission line and allows LVDS channels to operate with high signal frequency. However, LVDS interconnects suffer from a significant disadvantage due to their low ratio of information capacity to number of I/O pads. This makes LVDS extremely costly in I/O pads when used to construct SLI channels.



LVDS channels

- Current-mode drivers
- Data encoded as
  - {01} or {10}
- Advantages
  - Low switching noise
  - Large GDP
  - Common-mode noise rejection
  - EM coupled transmission lines
  - Low noise => low voltage swing
- Disadvantages
  - Wasteful in I/O pads



MBDS channel

- Scale up LVDS driver
- Data encoded such that exactly half the bits in each symbol are 1
- "N-choose-M (nCm)" symbols
  - {0011}, {0101}, {0110}, {1001}, {1010}, {1100}
- Advantages
  - Same transmission characteristics as differential
  - Higher information capacity

Figure 3.1: Side-by-side comparison of LVDS and MBDS channels.

3.2. Multi-Bit Differential Signaling (MBDS)

Multi-Bit Differential Signaling (MDBS) [88, 89, 90] is a new technology for building SLI channels. MBDS channels retain the favorable noise immunity and transmission characteristics of LVDS channels but have higher information density. This property allows MBDS to make more efficient use of I/O pad and power resources. In an MBDS channel, a standard current-mode LVDS driver design is scaled up such that it drives more than two wired outputs. The only restriction is that this number must be an even value. 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 common reference point for the receivers. The termination network guarantees that the voltage of the common point will always be the average of the voltages at each point on the termination network. 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 therefore held constant at the DC offset of the driver. This technique is effectively a physical-layer *channel code*, and is called "N choose M" encoding. One LVDS receiver is used for each of the wires forming the channel. The LVDS receiver senses the voltage of each point on the termination network relative to the common point. As in an LVDS channel, a positive or negative voltage is conditioned to CMOS levels representing a binary value.

A comparison of LVDS and MBDS channels is shown in Figure 3.1. MBDS channels share the high-performance noise rejection and transmission characteristics of differential drivers through their use of current-steering drivers, coupled transmission lines, and differential reception. However, MBDS channels have higher information capacity than LVDS channels, because an MBDS channel carries more data than a set of differential channels consisting of the same number of wires. Another way to view this is that MBDS channels use fewer I/O pins than a differential interconnect when required to carry the same amount of data. As a side effect, MDBS interconnects require fewer channels acting as current sources (corresponding to fewer encoded 1-bits) than differential interconnects having the same data capacity, making MBDS interconnects more power efficient. An additional benefit of MBDS is that the nCm channel code has other useful properties, such as an inherent error detecting capability and unused symbols.

3.3. "N choose M" (nCm) Encoding and Support for Additional Coding Features

MBDS channels utilize a physical-layer channel code called "N choose M (nCm)" encoding in order to guarantee that the voltage of the common point remains fixed. This is achieved by

restricting the data that may be sent over an MBDS channel to a set of predefined symbols, where each symbol contains a fixed number of 1-bits. In order to maximize the size of both the symbol set and the signal margins, the number of 1-bits in each symbol is always half of the channel width. In other words, such a channel is comprised of *n* wires, where *m* of these wires are always transmitting 1-bits and *n-m* wires are always transmitting 0-bits. For example, a 4 choose 2 (4c2) channel uses four wires such that at any time exactly two of the wires will be energized with 1-bits. When compared to the same wires configured as two LVDS channels, the 4C2 channel has roughly 25% greater information capacity. This is because the 4C2 channel can transmit any of the symbols in the set {0011, 0101, 0110, 1001, 1010, 1100} while the two differential channels are restricted to the symbols {0101, 1010, 0110, 1001}. This corresponds to approximately 2.58 bits of capacity for the 4c2 channel versus 2 bits for the LVDS channels.

Consider the set X_{nm} such that X_{nm} is the set of all valid code symbol encodings in an nCm channel. The size of X_{nm}, which is the number of available code symbols, is computed as:

$$\phi\{X_{nm}\} = \frac{n!}{(n-m)!\,m!}$$

Regardless of the number of symbols allowed with a particular interconnect configuration, each symbol must be mapped to a binary data value at the inputs and outputs of the channel. Since incoming and outgoing data will always be an integral number of bits, the effective bit width, bit_{eff}, defining the number of bits coming into and out of the channel before encoding and after decoding is computed as:

$$bit_{eff} = floor(log2(\phi\{Xmn\})$$

Using effective bit width as a metric, Table 2.1 compares the relative power consumption, pad count, and symbol utilization for several MBDS channel configurations relative to a differential

interconnect that carries an equivalent number of bits. Each wire in a channel that sends a 1-bit as part of a symbol acts as a current source and dissipates energy across the termination network. As a result, the relative power consumption of the two configurations is computed as $P_{eff} = m / bit_{eff}$, the ratio of *m*, the number of wires energized to '1' in the MBDS channel, to bit_{eff}, the number of wires energized to '1' in an equivalent differential interconnect. The relative pad count is computed as $RP=n/(2* bit_{eff})$, the ratio of *n*, the number of wires in the MBDS channel to $2*bit_{eff}$, the number of differential wires required in an equivalent differential interconnect. From the data in the table it is clear that a 25-34% improvement in power efficiency and pad utilization can be achieved with relatively small values of *n*.

Table 3.1: "N choose M" symbol sets.

Channel type	Available code symbols	Effective bit width bit_{eff}	Relative power consumption $P_{eff} = m/bit_{eff}$	Relative pad count $RP=n/(2*bit_{eff})$	% symbol utilization ut=2^{biteff}/cw	Raw code rate $rate=bit_{eff}/n$
2C1	2	1	100%	100%	100%	50%
4C2	6	2	100%	100%	66%	50%
6C3	20	4	75%	75%	80%	67%
8C4	70	6	66%	66%	91%	75%
10C5	252	7	71%	71%	51%	70%
12C6	924	9	66%	66%	55%	75%
14C7	3432	11	63%	63%	59%	79%
16C8	12870	13	61%	61%	63%	81%
32C16	601080390	29	55%	55%	89%	91%
36C18	9075135300	33	54%	54%	94%	92%

The nCm symbol set provides unused symbols as a result of mapping binary values into nCm symbols. The number of unused symbol for each nCm code set is shown in Table 1 under '% symbol utilization." This value is computed as $2^{biteff}/\phi(X_{nm})$, or the largest power of two fewer than the number of symbols divided by total size of the code set.

Valid nCm symbols only represent a subset of possible *n*-bit binary values. This is represented in the table as "raw code rate," which is computed as the bit capacity of the channel divided by the number of wires it requires. As a result, nCm symbol sets have a built-in ability to detect many types of channel errors. An nCm receiver may detect many types of channel errors if the result of the error yields a symbol that is not a valid nCm symbol.

In this work, the unused symbol space and inherent error detection properties of the nCm symbol set is used to build a new class of error control codes that requires lower information overhead relative to traditional error control codes. This class of error control codes is designed to correct bit errors that occur within nCm symbols. In this scenario, single or multiple signal errors are caused when, during the instant of sampling at the differential receiver, the relative voltage difference between any particular physical connection within the channel and the channel's corresponding common point does not exceed the minimum threshold voltage for that wire's receiver. Signal conditioning on the output of the differential receivers will induce a stable CMOS value in this situation. However that resulting symbol value may be in error. Even single bit errors occurring within nCm symbols will prevent the entire corresponding binary-mapped message from being decoded. This makes even single-bit correction within an nCm symbol have a profound effect, as this is equivalent to multi-bit correction relative to the binary source data that is transmitted across the channel.

4. PREVIOUS WORK

This chapter provides an overview of previous work in four areas that are relevant to the work presented in this dissertation. The first section is a brief history and survey of recent work in error control code theory. The second section is a survey of system-level interconnect technology that is currently employed or proposed for state-of-the-art and next-generation systems. We examine this technology from a circuit and data encoding standpoint. The third section is a survey of recent work in high-performance system-level interconnect technologies Each of these projects employs a form of data encoding or signal modulation to increase signal integrity, reduce I/O pads, or reduce power. The last section is a brief survey of recent work in high-performance optical interconnect technologies.

4.1. Error Control Coding

The field of code theory began in 1948 when Claude E. Shannon [37] wrote a landmark paper that showed that proper encoding of data can reduce errors in a noisy channel or storage medium. Shannon also derived the famous Shannon Limit, which defines the maximum amount of error-free information that may be transmitted through a single-bit-serial channel given the channel's bandwidth and its signal-to-noise ratio (assuming the only channel noise source is additive white Gaussian noise). This limit is defined as:

$$C = BW * \log_2(1 + S/N),$$

where C is capacity in bits per second (inclusive of error control coding), BW is channel bandwidth in hertz, and S/N is signal-to-noise ratio (power ratio -- not expressed in decibels). Since then, there has been continuous work to develop new codes and efficient encoding and decoding mechanisms in order to develop a communication channel that meets the Shannon Limit. In this section, we will review the relevant research that has made error correction coding possible. We then describe several relevant research projects in this field.

Error Control Codes (ECC) are used in environments where noise affecting a communication channel causes the data carried by the channel to be subject to random reception errors. The goal of an error control code is to detect and recover from such errors. ECC is a mechanism for encoding information using more information symbols than are necessary to convey the raw information [84]. As a result, the code space of an ECC code, which represents every possible valid encoding, or *code word*, only represents a subset of the possible binary representations given the width of the code word. This is why code spaces are also referred to as subspaces. Errors occurring within a received code word will yield a bit pattern that exists outside the code space and represents an invalid encoding. The receiver performs correction by determining the most likely code word that was originally transmitted. This code word is assumed to be the member of the code set that most closely resembles the received bit pattern.

In 1950, Hamming [43] introduced the theory of linear codes. These codes are defined as (*n*, *k*, *d*, *q*) codes, where raw information at the information source is divided into segments of length *k* and encoded into a code word of length *n*. The code rate of such a code, which defines it *overhead* or *efficiency*, is defined as *k* / *n*. All linear codes have a Hamming distance *d* that

defines its maximum binary error detection (d-1) and error correction ((d-1)/2) ability. The symbols that form the elements of the code word are values of base-q (i.e. for a binary code, q = 2). In block codes, error correction operates at symbol granularity, meaning that any symbol that contains bit errors may be corrected regardless of how many bit errors are present within the symbol, assuming there aren't more symbol errors than the code is capable of correcting as defined by its distance.

Linear codes are defined as vector subspaces formed by a set of basis vectors, referred to as a *generator matrix*. In other words, linear codes have a code space (every possible code word) that may be determined from multiplying a message space (every possible value of width k) by a generator matrix. The code space of Lightweight Hierarchical Error Control Codes cannot be determined in this way. This makes these codes *non-linear*. Every linear code is defined by its generator matrix and is also associated with a *parity-check matrix*, which is a simple transformation of the generator matrix and is used for detecting and correcting errors in the code word (if the generator matrix is defined as $[I^k \; A^{k \; x \; (n-k)}]$, the parity-check matrix is defined as $[-A \; I^{n-k}]$ modulo q). In order to encode linear codes, k digits of raw data, represented by a 1 x k information vector, are multiplied by the k x n generator matrix to yield a 1 x n code word. In order to decode linear codes, the receiver multiplies the 1 x n code vector by the transpose of the ($n - k$) x n parity-check matrix to yield a 1 x ($n - k$) syndrome. The syndrome is computed as a zero vector if no transmission errors have occurred within the code word. If transmission errors have occurred, the syndrome is used to determine the error vector, a vector that has effectively been added (in modulo-q) to the code word to produce the error(s). This is possible because the

syndrome of the received data is guaranteed to equal the syndrome of its corresponding error vector.

Linear codes may be systematic or non-systematic. In a systematic code, every code word consists of the original k, unencoded values along with $n - k$ additional values called parity digits. Retrieving the original k information bits from the codeword is trivial in a systematic code. Non-systematic codes have code words that bare no resemblance to the original unencoded information. Systematic and non-systematic codes are equivalent in their relationship between the number of parity symbols and the resultant distance of the code. Building good linear codes and designing efficient decoding algorithms/architectures forms the basis for most of the work in error control coding theory.

Consistent with these goals, several researchers [38, 39, 40, 41, 42, 44] have developed and analyzed methods of systematically constructing, encoding, and decoding block codes. Prange [45] introduced the theory of cyclic codes, a subclass of linear codes that allow efficient hardware encoding using shift registers. Several authors [46, 47, 48, 49, 50] have devised architectures and algorithms that efficiently decode several types of cyclic codes. BCH codes, which form a subclass of cyclic codes, were discovered by Hocquenghem in 1959 [51] and independently by Bose and Chaudhuri in 1960 [52]. BCH codes were the first codes that could be systematically constructed given a desired block length n and distance d. These codes were proven to have a cyclic structure by Peterson in 1960 [53]. Peterson also developed the first decoding algorithm for these codes. Gorenstein and Zierler [54] were the first to apply BCH codes to nonbinary base p^m symbols, where p is a prime number. Further work in decoding

algorithms for BCH codes was performed by Gorenstein and Zierler [54], Chien [55], Forney [56], Berlekamp [57, 58], Massey [59, 60], and Burton [61]. Reed and Solomon introduced the most widely used subclass of symbolic BCH codes in 1960 [62]. Reed-Solomon codes are simple to construct and are widely used for error control in magnetic/optical disks as well as several HDTV transmission standards. In most applications, bits from independent blocks are interlaced such that long burst errors are spread across multiple blocks and are thus less likely to corrupt more symbols than can be corrected. Reed and Solomon's most important result is the proof that their subclass of BCH codes meets the Singleton bound. This means that these codes provided the highest possible error correction capability (highest distance) for valid values of n, k, and q. In other words, Reed-Solomon Codes are maximum distance separable (MDS) codes, which are provably optimal relative to power and code rate.

For cyclic codes whose data symbols conform to a Galois Field (meaning that the symbol base that is a prime or a power of a prime, such as in binary codes, BCH codes and Reed-Solomon codes), encoding is equivalent to representing the k symbols of source data and the generator matrix as polynomials. The coefficients of these polynomials are elements of the Galois Field. The coefficients of the message polynomial are the message symbols themselves. Encoding is equivalent to multiplying these two polynomials modulo-q (using polynomial convolution). The resultant polynomial's coefficients represent the code word. For a systematic ECC code, the parity digits may be computed by themselves by computing the remainder when dividing a shifted message polynomial by the generator polynomial. In hardware, encoding may be performed by table lookup, but this type of encoding is usually not feasible for large code word sizes. Encoding may also be performed using shift registers and XOR gates (forming a linear

feedback shift register). However, hardware encoding in this way implies relatively high latency, requiring a clock cycle for each symbol in the width of the code word.

Any code that meets the Singleton bound (which includes Reed-Solomon codes) is referred to as a maximum distance separable (MDS) code [84]. MDS codes have basic restrictions when applied as a symbolic ECC code. Such a code must be applied over base-p^m symbols and is limited to a block length of p^m. Seroussi and Roth [32] developed extensions to MDS codes that allow the maximum block size to reach $p^m + 1$. De Boer [24] and Faldum and Willems [23] have discovered a class of codes called "almost-MDS codes" or "near-MDS codes" that allow the maximum block size to reach $p^m+1 - 1$ at the cost of at least one additional parity symbol (thereby reducing code rate).

Convolutional codes, introduced by Elias [79] in 1955, form another subclass of binary linear codes. Convolutional codes are defined as (n, k, m) codes, and differ from block codes in that the encoder and decoder contain memory. The n encoder outputs at any given time depend on the k inputs as well as the previous m input blocks. Convolutional codes are implemented as a linear sequential circuit and are typically used with small values of k and n and a large input memory m. Convolutional codes are also defined by their constraint length, defined by $n * (m + 1)$, which defines the maximum number of encoder outputs that can be effected by a single information bit. There is no known mechanism for constructing convolutional codes with a desired distance property. Convolutional codes must be constructed using computer search, restricting effective convolutional codes to small constraint lengths. Convolutional codes are used in extremely noisy channels where high reliability is needed and low code rate is

acceptable. These codes are used in various radio applications such as CDMA/GSM cellular networks, 802.11 wireless ethernet, satellites, and deep-space probes. Decoding convolutional codes is typically performed by a Viterbi decoder [80], which is implemented as a finite state machine (sometimes represented by a “Trellis diagram”).

Forney [65] was the first to propose the use of concatenated codes, which is a method of encoding data using multiple error control codes, of potentially different types/classes. When using a concatenated code, binary data is broken into a number of multi-bit symbols. A symbolic ECC code, called the “outer code”, is applied and parity symbols are computed. After parity symbols are added to the block, a binary ECC mechanism, called the “inner code”, is applied over each symbol. This causes parity bits to be appended to each symbol in the block including the outer code’s parity symbols. This method is used to decode extremely long codes with less decoding hardware. In other cases, data is first encoded with a block code (such as a Reed-Solomon code), then the encoded block is again encoded with a convolutional code prior to transmission. Concatenated convolutional codes used with bit interleaving techniques and iterative decoding form the basis of the popular Turbo Codes [67] that are seeing widespread use in modern radio (wireless) communication systems.

Error correction codes also have the ability to correct special types of errors such as erasures. An erasure is a symbol within a code word that is known by the receiver to be in error. Luby et al [63] has constructed an efficient decoding mechanism for correcting erasures in certain types of symbolic block codes. Seroussi and Roth [64] have developed codes called “location correcting

codes" that can correct symbols where the error value is known rather than the error location (as with an erasure).

"N choose M (nCm)" codes, the same codes used in Multi-Bit Differential Signaling and sometimes referred to as "m-out-of-n codes," "balanced codes," and "constant weight codes," have been previously used for error detection and self-checking circuits. Ramabadran [66] has proposed using nCm codes for perfect error detection in asymmetric channels, where bit errors are modeled as either a 0-to-1 flip or a 1-to-0 flip but not both. In such an error model, any bit error would result in an invalid code word, making nCm codes highly effective in this case. There has also been much work [73, 74] in computing the probability of an undetected error in binary symmetric systems that use nCm codes along with an automatic repeat-request (ARQ) mechanism. Several researchers have developed self-checking circuits and logic gates, called Totally Self-Checking Checker (TSC) circuits, through the use of nCm encoding [69, 70, 71, 72].

Szymanski [68] has proposed a forward error correcting mechanism for short-distance VCSEL-based optical interconnects that uses a form of concatenated codes along with an automatic repeat-request (ARQ) mechanism. In this case, an error-detecting cyclic-redundancy-check (CRC) code is used for the outer code and a short binary BCH code forms the inner code. A CRC code is a simple way to add parity bits through a hash computation to long binary strings and allows for detection for some types of multiple bit errors.

4.2. Survey of State-of-the-Art Interconnect Technology

This section examines several current examples of state-of-the-art system-level interconnect technologies and standards for next generation system-level interconnect technologies. First, it

examines the technology used by Intel and Rambus, who currently manufacture products with high-performance processor-to-memory interconnects. These technologies use a hybrid form of single-ended (one wire per bit) and differential (two wires per bit) techniques. Next it examines Hypertransport and RapidIO, which are two competing interconnect standards for high-performance chip-to-chip and backplane applications. Both of these standards use differential signaling techniques along with additional data encoding as the physical-layer component for their interconnect standard.

The Pentium IV processor chip, manufactured by Intel Corporation, is a widely-used microprocessor for desktop computing [8]. This processor, manufactured in a 90 nm process, operates with a 3.4 GHz internal clock speed but is limited to an off-chip system bus speed of 800 MHz. This clearly illustrates the growing gap between the transmission rates of on-chip and off-chip (system-level) channels in modern systems, when compared to the original Pentium architecture that has a core speed of 100 MHz and a system bus speed of 66 MHz. The Pentium IV is packaged in a 478-pin flip-chip pin grid array package, of which 473 pins are used by the die. Of these, 85 pins are used for power and 179 are used for ground, accounting for 56% of the total pinout. This chip has an extremely high level of integration (~100 million transistors) and thus requires a large portion of its pinout to be dedicated to power and ground connections. Future chips that have higher levels of integration will have greater power demands and require an even larger portion of their pinout to be dedicated to power and ground connections. This leaves even fewer pins available for signaling through ultra-wide parallel I/O. This trend indicates that Intel may adopt a serial signaling technology for the system bus interface. The chip accepts a differential signal for its internal clock, because this signal is generated off-chip

and must be capable of running well into the GHz range. The system bus operates in the MHz range and is a highly parallel interconnect using "quasi-differential" signaling. Quasi-differential signaling is essentially a single-ended "bit-per-wire" technique; however, instead of using ground as its signal reference, an explicit off-chip reference is used for all channels in the interconnect. Four signal reference pins are located on the four inner corners of the PGA packages. The system bus is made up of a 33-channel unidirectional address bus and a 64-channel bidirectional data bus. Both the address and data busses are divided into 16-channel segments that may selectively enabled. The supply voltage is user-specified and is typically set at around 1 V, and the signal swing for the system address and data channels is specified as ref +- 10% * Vcc. This means that the signaling operates +- 100 mV relative to the reference points for a total single swing of 200 mV. Each 16-channel segment of the address and data busses uses a single parity signal that is used for single-bit error detection. This represents a relatively primitive form of error control coding as there is no mechanism in place for error recovery or request for data retransmission (signal errors would generate processor interrupts leading to a system halt). The chip offers signal termination on die, and high-frequency decoupling capacitors on the package.

Another company that designs high-speed chip-to-chip interconnects is Rambus Corporation. They have pioneered a serial bidirectional interconnect technology, called Rambus Signaling Levels (RSL), which is used for their high-performance memory systems [35]. RSL and its updated counterpart, QRSL, are both quasi-differential signaling mechanisms based on explicitly referenced channels. RSL is very similar to the signaling technology used in Intel's processor chip and thus has comparable transmission rate limitations. Rambus's most advanced

interconnect technology, called QRSL (Quad Rambus Signaling Levels), uses current-mode drivers to encode two bits on each channel by associating four unique voltage values across a 40-ohm termination. The four voltage ranges are recognized by a receiver that compares the input voltage relative to three fixed reference voltages. The set of voltage values, corresponding to each possible two-bit value, has a separation of 270 mV. This signaling technology is a variation on 4-Level Pulse Amplitude Modulation (4-PAM), discussed below. Rambus has developed and tested 2 Gbps signals using this technology. Rambus also offers a separate signaling technology standard, called SerDes (Quad Serializer/Deserializer) that is intended for chip-to-chip and backplane applications. This is a fully differential interconnect and uses 20 mA current-mode drivers over a 50-ohm terminated connection. This interconnect offers clock extraction using 8-bit / 10-bit Manchester data encoding for maximizing bit transitions for clock extraction. It is designed to run over transmission lines of up to 30 inches. Interconnects using this technology have been tested at 3.125 Gbs.

RapidIO and Hypertransport are two competing standards for next-generation system-level interconnect [4]. These technologies seek to implement high-speed serial interconnects rather than follow the longstanding trend of increasing interconnect bandwidth by designing increasingly parallel interconnects. Both of these standards are competing for widespread acceptance, both are currently being updated, and both seek to reach practical implementation at 2 Gbps. RapidIO uses the Low Voltage Differential Signaling (LVDS) [2] standard for its physical-layer, while Hypertransport uses Lightning Data Transfer (LDT) for its physical-layer. LVDS and LDT are both pair-wise differential standards with slightly different specifications (LDT is more power-aware than LVDS). Both RapidIO and Hypertransport are packet-switched

protocols at the network-layer, and specify a standard for inter-chip and backplane interconnect. In addition, both standards employ primitive coding mechanisms to prevent data and/or clock loss. RapidIO uses an 8 bit / 10 bit encoding for clock extraction and error detection, while Hypertransport uses CRC over 512 data units for error detection. These approaches do not offer channel error correction ability at the link-level, but offer retransmission mechanisms if a data or clock fault is detected.

4.3. Current Research in High-Performance Interconnect Technology

Recent work in high-performance interconnect technology is concentrated on serial interconnects that employ advanced circuit designs along with signal modulation and data encoding for pin and power reduction techniques. This section reviews several articles that characterize the problems governing high-speed interconnect technology. Next, it discusses and evaluates specific projects aimed at overcoming these problems through circuit and data encoding or modulation techniques. These projects are representative of current trends in system-level interconnect technology and provide a basis with which to compare the work presented in this dissertation.

An article written by Mark Horowitz and his group from Stanford University [3] is an exhaustive study on what factors contribute to signaling latency and throughput in device-level interconnects. This paper describes the effect of RLC parameters on driver and receiver circuits, modulation techniques, as well as considerations for clock extraction, signal referencing, and jitter. He concludes that next-generation system-level interconnect technologies should employ high-speed serial interconnects that include data encoding or signal modulation to address these problems. He proposes pulse amplitude modulation as a possible solution. Details on this technology are explained later in this chapter. In a similar article, [1], John Poulton from the

University of North Carolina at Chapel Hill concludes that narrow, high-speed serial interconnects, along with data encoding techniques are the keys to high-speed chip-to-chip interconnect.

Following similar motivations as the work presented in this dissertation, the Mark Horowitz's group [5] and a group from the University of Toronto [29] use signal modulation for reducing pin count, increasing code rate, and increasing signal integrity by optimizing the frequency characteristics for high-performance signaling. Both groups are using variations on 4-PAM (4-level pulse amplitude modulation), a multi-level signal modulation scheme similar to Rambus's QRSL technology. The basic idea of this modulation scheme is to switch from a digital "square-wave" signal to a modulated signal where pulses are sent with n different amplitudes, each representing $\log_2 n$ bits of data. The set of possible values that may be sent across this interconnect is referred to as a constellation. This modulation technique is very similar in concept to audio modulation applied to increase the throughput of telecom-based modems in the 1980's and early 1990's. The Stanford group also proposes driver pre-emphasis techniques, receiver equalization techniques, and phase-multiplexing of a serial channel.

Lightweight Hierarchical Error Control Codes exploit hidden information that is inherent in channels that have a code rate of less than 1, meaning that each wire forming the interconnect carries less than one effective bit. The source of this information is the unused, or redundant, portion of the channel capacity, and it is used to recover additional code rate that is lost when error control is added to the channel. 4-PAM channels offer high code rate by modulating 2 bits onto a single wire. As such, there is no hidden redundancy to exploit and therefore it is not

possible to employ similar techniques to achieve low-overhead error control for 4-PAM channels.

Another project from Mark Horowitz's Stanford group [6] proposes a solution to reduce I/O pin count and describes their work on a 2.4 Gbs interconnect. Their solution for reducing I/O pin count on parallel single-ended interconnects is to make the channels simultaneously bidirectional, where signals in both directions are superimposed on the same physical wire. Simultaneous bidirectional interconnects effectively double the interconnect between two chips while using the same number of wires. The receiver in each transceiver subtracts its own transmit signal from the line voltage to generate the receive signal. They note that coupling the transmit signal to the receive signal creates a number of extra noise sources that makes noise filtering and data synchronization an important issue in this design. They propose an aggressive receiver design that uses current integration techniques to filter out high-frequency noise on the receiver reference (low-pass filter). To synchronize data in the channel, both chips generate clock signals running in opposite directions. Each chip's incoming clock signal is used along with a phase-locked-loop to generate a receive clock that is delayed by a half-bit time in order to minimize the effects of skew and jitter. This technology does reduce pin count and provide a synchronization mechanism but introduces additional noise sources that must be handled by complex analog circuitry. Very similar work in simultaneous bidirectional interconnects was performed by the IBM Server Group in [7]. Unlike the Stanford design, they designed bidirectional differential interconnects. The receivers work by having the ability to measure four unique voltages across the termination resistors. This voltage is determined by the single-bit transmit state of both chips in the interconnect. Each chip, with knowledge of what signal its

driver is sending on the interconnect, can use the termination voltage at its receiver to determine its receive signal from other transmitting chip. Once again, while this technology serves to reduce pin count, the price paid is in relatively complex analog receiver comparator circuits.

Despite the high I/O requirement of differential signaling, there are several research projects that seek to design differential interconnects that support ultra-high transmission rates. In [2], Stefan Hirsch and Hans-Jörg Pfeiderer present and characterize several high-speed differential receiver circuits. Motorola [21] reports on several high-speed differential (LVDS) drivers and receivers implemented using silicon-on-insulator technology. These interconnects are fundamentally limited by their low absolute code rate.

Although differential channels are formed by two wires, these wires are received as a differential pair and therefore the driver input and receiver output corresponding to each two-wire channel is a single bit. In other words, a single bit enters and exits the end-points of a differential channel, formed by two wires. However, in multi-bit differential channels, a channel of width *n* produces *n* distinct binary outputs at the receiver. If no reception errors have occurred, this sequence of bits will match a valid nCm symbol. However, when errors affect the channel, it is possible for the receivers to produce any arbitrary binary sequence of width *n*. The fact that a binary output is produced for each wire in an MBDS channel is what facilitates the use of Lightweight Hierarchical Error Control Codes, and allows individual bits within the symbol to be corrected. This type of symbol correction is not possible with pair-wise differential signaling because the receiver outputs do not form a symbol set that can be partitioned.

Several groups are investigating data encoding mechanisms for reduction of signal frequency, handling data synchronization, or reducing power. A group from the University of Ferrara, Italy [30] utilizes an encoding called minimum run-length guaranteed codes (MRLG). This type of encoding guarantees that single, isolated bits do not occur in a bit stream and ideally allows encoding data with as much as twice the bit rate of the wire that is it transmitted on. This encoding works by replacing isolated bits in the source data with special symbols that do not contain isolated bits. The efficiency of this code is highly dependent on the characteristics of the data being transmitted across the channel. They have also designed simple encoding and decoding logic. This encoding provides energy savings and increased signal integrity due to the overall reduction in signal frequency. A group from Cornell proposes a data encoding mechanism for a high-speed asynchronous serial interconnect [33]. They use high fan-in multiplexing drivers and high fan-out demultiplexing receivers to multiplex eight data streams through a single-bit serial channel. In this design, three wires connect a transmitting chip to a receiving chip. These three wires implement a single-bit serial channel. Both chips operate a three-state finite state machine. The next-state logic depends on which bit is currently being transmitted. In each state, a pulse sent along one of the three wires represents a one-bit and a pulse sent along another wire represents a zero-bit. This allows the receiving chip to extract timing information and data from the channel without the need for a high-speed clock signal. They also propose a way to recover from driver and receiver state machine desynchronization due to bit errors. Binary error detecting codes are used over data sent over their interconnect. If a bit error is detected, the driver and receiver reinitialize themselves and reset their current state through a resynchronization mechanism.

Shin et al propose data encoding methods for reducing power consumption by reducing the number of bit transitions across wide busses [75]. These codes are variants of the Bit-Invert method, where bus data is inverted prior to transmission if the distance between the current data and the previously transmitted data is greater than half the width of the bus. Stan et al proposes similar low power encodings over both time and space to reduce bus switching (time) and number of transmission lines driven high during any transmission period (space) [76].

4.4. Current Research in Non-Electrical Interconnect Technology

Optical interconnect has been proposed as a solution for solving many of the problems inherent in electrical interconnects. In optical interconnects, signals are typically transmitted and received orthogonal to the chip's top or bottom surface. This technology promises ultra-dense, low-latency, and energy-efficient 2-dimensional interconnect arrays that have reduced levels of capacitance, crosstalk, and reflections compared with electrical interconnect technology. Vertical cavity surface emitting laser (VCSEL) and photodetector technology has made optical chip-to-chip interconnects feasible, but effective packaging technology is required to make such interconnects practical. In this section, we review projects in optical chip-to-chip interconnect technologies.

Our own research group is involved in guided-wave optoelectronic interconnect research. One of our previous projects is the construction of an optoelectronic multi-chip module demonstrator system [9, 10, 11]. This system is built by directly coupling optoelectronic components to multiple surfaces of a rigid fiber image guide (FIG) [9] structure. The structure is built using multiple layers of FIG cut at various angles, which allows both arbitrary signaling paths and signal fanouts to be created among the chips mounted on each surface. Each optoelectronic

component is constructed from a transparent-substrate silicon-on-sapphire logic die that is flip-chip-bonded to an 8x8 vertical cavity surface emitting laser array and an 8x8 detector array, both having a 250 um pitch. The fiber image guide structure acts as both the structure and optical transmission medium for the multi-chip module. In this case, the goal is to provide extremely dense, highly parallel optical interconnects to provide high bandwidth, low latency interconnects between chips. We have also presented research on other mixed-technology, mixed-substrate, system-in-package technologies based on similar principals [12, 13]. There are several other groups working with fiber image guides for parallel, chip-to-chip optoelectronic interconnects. These include the NEC Research Institute, who integrate polymer FIGs directly into circuit boards to create optical signaling paths between chips [14, 15], and McGill University who has used flexible fiber image guides to create 10-channel parallel interconnects [16]. There has also been interest in creating optical interconnects using 2-dimensional fiber ribbon cables rather than fiber image guides [25].

Several groups are currently developing free-space optoelectronic interconnect technology. In this case, instead of using image guides to propagate optical signals, point-to-point optical interconnects are created among chips sharing a common circuit board and using microlenses to direct (refract) vertically fired lasers and using mirrors suspended above the circuit board to reflect those signals back to neighboring chips. Groups from UCSD [17] and from George Mason University [18] have built and tested several such chip-to-chip interconnects.

5. PROBLEM STATEMENT

The goal of this work is to develop a new error control code that may be efficiently applied over system-level interconnects. In order to achieve this goal, this code must fulfill three requirements. First, it must be *effective* at correcting the types of signal errors that occur within system-level interconnects. Second, it must be *viable* to integrate support for this code to the I/O interfaces of system-level interconnects. Third, the effectiveness of this code must be *verifiable* through experimentation.

This error control code must provide sufficient benefit in terms of increased signal integrity and noise immunity to justify its costs in information overhead. Ideally, it must achieve an optimal trade-off between information overhead and error correcting power based on the types of noise that affect system-level interconnects. It is wasteful to allocate information overhead to endow this code with the ability to correct types of errors that do not occur. For system-level interconnects, the dominant error sources are randomly distributed errors that originate from high-frequency operation of driver and receiver circuits, power supply noise, and crosstalk among near-by signal transmission lines.

This code must be designed such that it is possible to tightly integrate the required support logic into system-level interfaces. Examples of such interfaces include on-chip routing/switching elements and off-chip I/O pad-frames. Speed and area requirements for the encoder and decoder are two important considerations for this goal.

The most important requirement for the encoders and decoders is operating speed. The encoders and decoders must not be the speed-limiting component to their corresponding channel interfaces. This means they must operate at the same data rate as the channels for which they are designed. In this work, it is assumed that these data rates may be high enough to match system core logic speeds. These speeds stretch well into the multiple gigahertz range for state-of-the-art CMOS fabrication processes. The interconnects described in this work have multiple-bit capacity, meaning that effective aggregate interconnect throughput, which is signaling frequency multiplied by interconnect capacity, may stretch into the hundreds of gigabits per second range.

In addition to the speed requirement, the encoders and decoders have an area requirement. Only a small fraction of total chip real estate is reserved for system-level interfaces. As such, the total chip area required to implement each encoder and decoder should be small enough to fit within the chip area that is normally allocated for these interfaces. In order to achieve this requirement, it must be possible to implement the encoding and decoding components in a small area using a small number of standard logic gates and latches (i.e. a few hundred).

Both the area and speed requirements may be achieved by utilizing a block code with a small block width and keeping the relative information overhead as low as possible. In addition, structural features of the error control code may be used to further reduce encoding and decoding complexity.

In order to verify this technique, several LHECC configurations must be constructed that have variable interconnect widths, data capacities, and error correcting power. Each of these configurations has corresponding requirements for information overhead. Afterwards, each of these codes must be simulated against accurate interconnect and noise models to determine the optimal trade-off point for maximum code effectiveness and maximum effective interconnect data throughput.

6. APPROACH

Lightweight Hierarchical Error Control Codes (LHECC) are a new class of error control codes that have properties that make them practical, efficient, and viable for use over system-level interconnects. LHECC relies on a hierarchical approach to combine symbolic and binary encoding techniques. This approach allows it to take advantage of inherent properties of the underlying physical-layer channel code. These properties are used to minimize both information overhead and encoding and decoding complexity. This chapter begins by discussing the inherent properties of the channel code that are used in the LHECC approach. Next, it provides a theoretical study of LHECC including how these codes are encoded and decoded. Finally, it describes example encoder and decoder architectures that are appropriate for system-level I/O interfaces.

6.1. The Bottom-Layer: MBDS "N choose M (nCm)" Channel Code

The "N choose M (nCm)" physical-layer channel code used by Multi-Bit Differential Signaling (MBDS), as described in Chapter 3, offers two properties that are used to facilitate the LHECC technique. The first property is an inherent error detection capability that allows the receiver to detect certain types of bit errors within any received nCm symbol. This property is used to reduce the number of parity symbols required to achieve error control in a block code. The second property is the existence of leftover nCm symbols when nCm symbol sets are mapped to binary messages. These leftover symbols are used in several ways to offset the overhead required by each parity symbol.

6.1.1. Inherent Error Detection Capability of nCm Symbol Sets

By definition, every nCm symbol set has an inherent ability to detect certain types of bit errors that occur during transmission. An nCm symbol containing bit errors will be detected if the symbol sampled by the receiver does not contain an appropriate number of one-bits as defined by half the symbol width (the *m* parameter of the symbol set). As a result, any random occurrence of bit errors 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 combination of an odd number of bit errors can be detected, while only a subset of combinations of even numbers of bit errors can be detected. Figure 6.1 illustrates this property by showing the effects of changing random bits within a 4c2 symbol. When any one- or three-bit error occurs within this symbol, the resultant four-bit sequence does not match a valid 4c2 symbol encoding. However, only certain types of two-bit errors yield an invalid 4c2 symbol encoding.

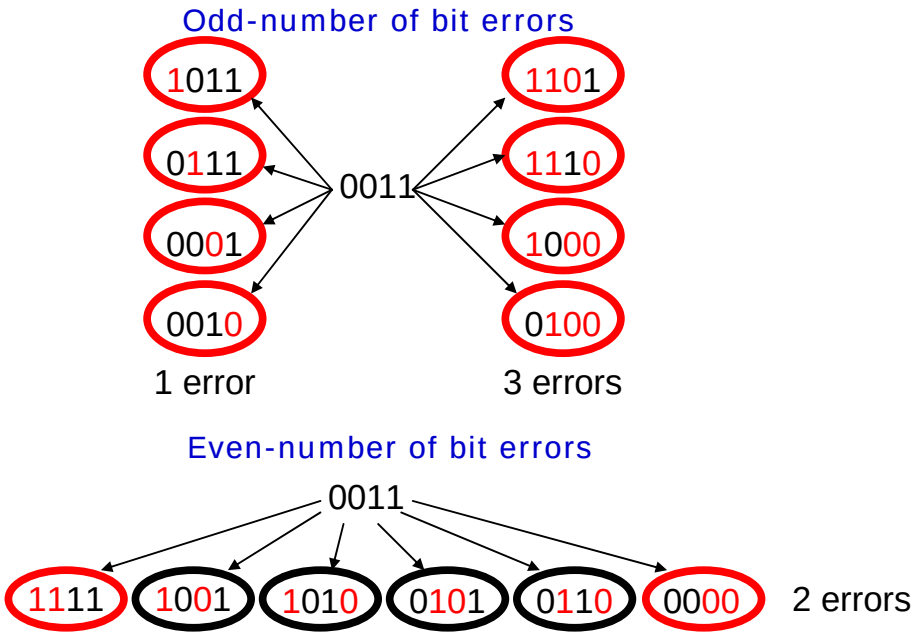


Figure 6.1: Inherent error detection in nCm symbol sets. This figure shows how various bit errors occurring during transmission of a 4c2 nCm symbol may be detected as an invalid nCm symbols at the receiver.

The probability to detect errors within an nCm symbol containing an even number of bit errors can be determined in the following manner. Assume an nCm symbol is received that contains *e* bit errors where *e* is an even number. Assume that these errors are represented as a vector of length *n* called an error vector. The error vector is effectively XOR'ed to the original transmitted symbol yielding the received symbol. Therefore, the error vector contains one-bits corresponding to each bit that is changed in the original symbol. For example, an error vector of [1 0 0 1] indicates that the most significant and least significant bits in the symbol were changed (flipped) during transmission. There are exactly *e* 1-bits in the error vector representing the bits that are changed in the original symbol. There are *n* bits in the original symbol, so there are C(*n*,*e*) unique error vectors.

Table 6.1: Probability for detecting an even number of bit errors for various nCm symbol sets.

Chan	P(detect 2-bit error)	P(detect 4-bit error)	P(detect 6-bit error)	P(detect 8-bit error)
4c2	33%			
6c3	40%	40%		
8c4	43%	49%	43%	
10c5	56%	48%	48%	56%

The receiver will not detect errors in an nCm symbol if the errors result in another valid nCm symbol. Table 6.1 shows what portion of even numbered bit-errors are detectable by several nCm symbol sets. This occurs if an equal number of 1-bits to 0-bits are changed in the originally transmitted symbol. There are m one-bits and $n - m$ zero bits in the original symbol, so only $C(m,e/2) * C(n\text{-}m,e/2)$ possible error vectors will yield a valid symbol. Therefore, given e, $1 - ([C(m,e/2) * C(n\text{-}m,e/2)] / C(n,e))$ defines the portion of even-numbered bit errors that can be detected by a particular nCm symbol set. For example, if 2 bit errors occur within a 4c2 symbol, there are $C(4,2) = 6$ unique error vectors, and $C(2,1) * C(2,1) = 4$ possible error vectors that yield a valid 4c2 symbol. Therefore, $1 - (4 / 6) \sim= 33\%$ of cases where two bits are flipped will yield invalid symbols and will be detected by the receiver.

6.1.2. Excess Symbols Available in nCm Symbol Sets

The size of any nCm symbol set where the number 1-bits in each symbol (m) is equal to half the symbol width will not span an integral number of bits. In other words, the size of the symbol set will not equal a power of 2. Therefore, given any arbitrary one-to-one mapping between the symbols in an nCm symbol set and binary messages, there will always be symbols that remain unmapped.

For example, consider the 4c2 symbol set shown in Figure 6.2. A mapping must be established between each 4c2 symbol and a binary message. While the size of this symbol set is 6, the number of symbols that are mapped must span an integral number of bits, e.g. floor ($\log_2 6$) = 2. For 2 bits, four 4c2 symbols are mapped to two-bit binary messages, leaving two symbols unmapped and potentially unused.

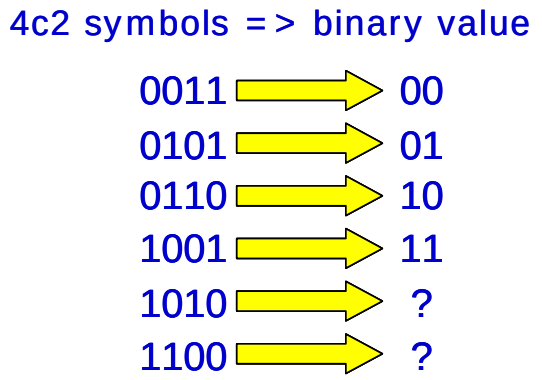


Figure 6.2: Unmapped symbols in a 4c2 symbol set.

One way to exploit excess symbols is to build interconnects using multiple MBDS channels in parallel. The absolute code rate of the interconnect, or the bit capacity divided by the number of wires in the interconnect, increases not only with the channel width but also with the number of parallel channels used that form the interconnect. For example, consider the interconnect shown in Figure 6.3. A single 4c2 channel can carry one of 6 possible symbols, equivalent to floor($\log_2(6)$) = 2 bits. This yields an absolute code rate of 2 bits / 4 wires = 50%. However, two 4c2 channels in parallel is equivalent to floor($\log_2(6^2)$) = floor($\log_2(36)$) = 5 bits, yielding an absolute code rate of 5 bits / 8 wires, or 62.5%. In general, in order to increase the absolute code rate in this way requires *d* channels, where *d* equals the inverse of the "fractional" portion of the absolute code rate of an independent channel, or *d* = floor(1 / ($\log_2$(*symbol set size*) – floor($\log_2$(*symbol set size*)))). Therefore, for every *d* channels used to form the interconnect, the

total capacity is increased by one bit over the capacity otherwise achieved with independently mapped channels.

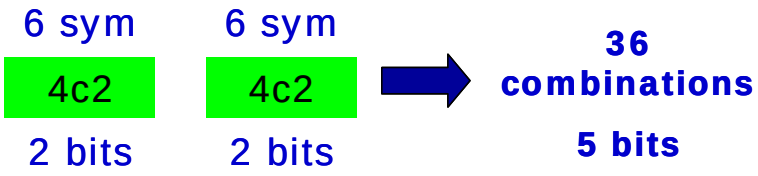


Figure 6.3: Using unmapped symbols to expand binary message space.

Another method for exploiting the unused symbols in an nCm symbol set is to simply consider a subset of the symbols "off-limits" for transmission. This option is favorable when certain symbols in the symbol set are counteractive to the goals of additional encoding features such as error control. This is explained further in Section 6.4.

6.2. Hierarchical Encoding and Decoding

Hierarchical encoding offers a methodology to exploit the inherent capability for binary error detection present in nCm symbol sets. The encoding process is shown in Figure 6.4. In this approach, binary source data at the transmitter are encoded to form a sequence of valid nCm symbols forming a *code word*. This sequence of symbols is used for physical transmission across parallel MBDS channels.

To encode an LHECC code word, binary source data is logically split into two portions. These portions are referred to as the s-data and c-data. The s-data portion consists of a sequence of symbols that are encoded into the *high-level code*. The high-level code is a traditional (n, k, d, q) linear block code. As such, one or more parity symbols are computed and appended to the s-data. This encoding forms the *high-level code block*. The resultant code block is used as a set of

guidelines for selecting the sequence of nCm symbols that are used to form the code word. The c-data portion consists of a sequence of symbols that are encoded into the *low-level code*. The low-level code is formed by selecting nCm symbols based on guidelines set by the high-level code block.

When applied to every possible value of the source data, this encoding technique forms a non-linear binary code space. This code space is non-linear because it cannot be formulated as a vector subspace (by multiplying a message space by a generator matrix). This code space has an arbitrary Hamming distance that is selected by the code designer. Also, it effectively encodes source data while restricting each symbol in the code word to a particular nCm code set. Most importantly, it achieves error control without requiring dedicated parity symbols that don't carry any source data. In addition, fewer parity symbols are required to correct nCm symbols in the code word as compared to a traditional error control code because most errors manifest themselves as invalid nCm symbols, which are detected at the receiver as symbol erasures instead of pure symbol errors. These attributes make the code lightweight.

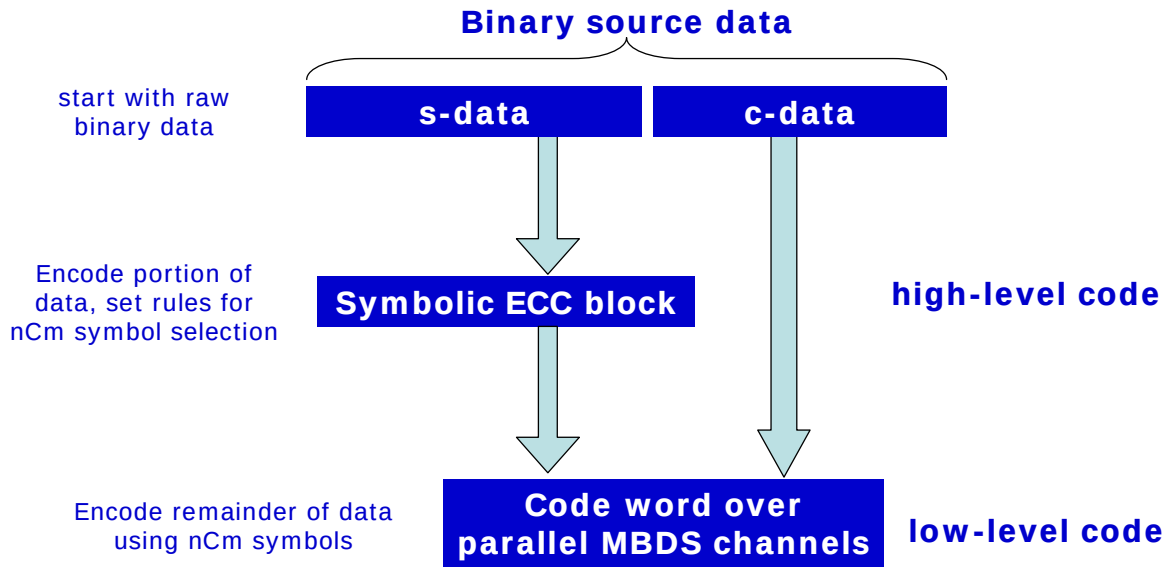


Figure 6.4: Hierarchical ECC encoding

6.3. Low-Level Code

The low-level code is a component of LHECC and relies on a method of transforming an nCm symbol set in such a way that its inherent error *detecting* capability is increased to become an error *correcting* capability. This operation requires *partitioning* a symbol set, and is the most important component to the LHECC approach.

Observe that any pair of randomly selected nCm symbols will differ in at least 2 bit positions. Therefore, the effective Hamming distance of nCm symbol sets is 2. The number of detectable errors for any code set is defined as $d - 1$, where d is the Hamming distance. The fact that nCm symbol sets have a Hamming distance of 2 is simply another way of stating that they have an inherent ability to detect single bit errors. However, the number of *correctable* bit errors of any code set is defined as 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. Therefore, the Hamming distance of an nCm symbol set must be increased in order to achieve a minimal level of error correction capability. This can be accomplished by partitioning the nCm symbol set into subsets such that the Hamming distance of each subset is defined by the distance parameter of the partitioning operation. This Hamming 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 a single-bit error within each nCm symbol in a subset while partitioning for distance 6 would allow correction of a one- or two-bit error within each nCm symbol in a subset. The only restriction of this partitioning operation is that the subsets must be of equal size as this is a requirement for the high-level code.

The partitioning operation forms the low-level code and is defined as a (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 subset Hamming distance. The subsets are enumerated such that each is assigned a unique base-s value that identifies the subset. Likewise, the nCm symbols within each subset are enumerated such that each is assigned a unique base-c value that identifies the symbol within its corresponding subset. Therefore, every nCm symbol is associated with a corresponding subset number, or s-data component, and symbol number, or c-data component. The s-data value and c-data value that are associated with each nCm symbol are an integral component to the encoding and decoding procedure.

The partitioning operation need not utilize the entire nCm symbol set, meaning that not every symbol need be assigned to a subset. In many cases, it is impossible to partition an nCm symbol set for certain target distances or number of subsets if every symbol must be assigned. In this case, ignoring certain symbols represents a way to utilize unused symbols in an nCm symbol set. However, in order to maximize relative code rate, the product of s and c must be maximized while keeping c maximal relative to s. This is explained further in section 6.11. In addition, encoding and decoding complexity are decreased when s and c are powers of 2 as this eliminates the need for base-conversion components in the encoder and decoder.

6.4. Partitioning Operation

Partitioning an nCm code set into equal-sized subsets for a given distance is equivalent to a graph coloring problem. To construct such a graph, each node represents a symbol in the nCm symbol set. Edges are formed between any pair of nodes whose symbols differ in number of

corresponding bit positions less than the chosen Hamming distance parameter. Graphs formed in this manner are highly dense and regular graphs. For example, if the target Hamming distance is 4, edges would connect each symbol to each other symbol that differs in only 2 bit positions. This would include all symbols differing in one 1-bit and one 0-bit, equaling C(m,1) * C(n-m,1) edges for each node. For example, for the colored graph with target distance 4 shown in Figure 6.5, each node is connected to 4 other nodes.

An example partitioning of a 6c3 symbol set into 4 subsets of 4 symbols each with a target distance of 4 is shown in Table 6.2. In this example, only 16 of the possible 20 symbols are used in the partitioning. An exhaustive search determined that it is impossible to partition a 6c3 symbol set with a target Hamming distance of 4 where all symbols are assigned to a subset. This is the case for many symbol sets when the partitioning distance is less than the symbol width. However, excluding nCm symbols in the partitioning operation represents another way that unused nCm symbols are exploited to facilitate construction of a LHECC code.

After a symbol set is partitioned, each subset is assigned a unique base-s value and each symbol is assigned a unique base-c value within each subset. These values define the s-data component and c-data component that is associated with each nCm symbol. For example, for the partitioning shown in Table 6.2, symbol 100101 is associated with s-data value 1 (because it is contained within subset 1) and c-data value 2 (because it is the third symbol within subset 1).

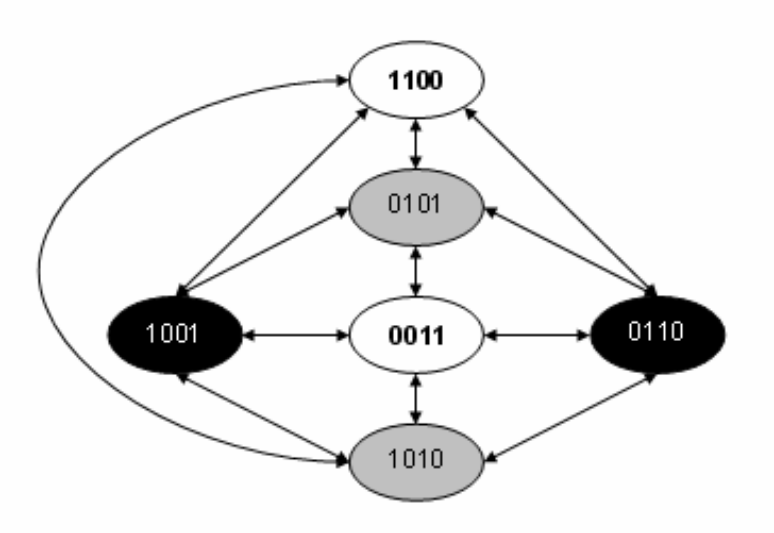


Figure 6.5: Graph coloring for 4c2 symbol set, distance=4.

Table 6.2: Example partitioning of a 6c3 symbol set with Hamming distance 4 into 4 subsets of 4 symbols each.

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

The search algorithm shown in Figure 6.6 is used to achieve symbol set partitionings. The algorithm requires that the code designer choose the nCm symbol set and the values of *s*, *c*, and d_l. The algorithm works by performing a depth-first search over the space of symbol subset assignments. This partitioning is performed offline during the time when an interconnect designer is designing the code. The results of the partitioning operation are encoded within the implementation of the LHECC encoder and decoder.

```
function partition (code_set, num_subsets, subset_size, distance), returns assign_state

initialize assign_state to empty
create empty stack of assignment states
push empty assignment state to stack

repeat until all subsets contain subset_size symbols {
    pop assign_state from stack
    find smallest subset number not currently assigned to subset_size symbols
    for each unassigned symbol in assign_state {
        if symbol fits into current subset (relative to its distance to all other members) {
            assign the symbol to subset
            push resultant state onto stack
            undo last assignment
        }
}

return assign_state
```

Figure 6.6: Partitioning algorithm.

6.5. High-Level Code

The high-level code of LHECC is a traditional symbolic linear block code. This block code is defined as a (n, k, d_h, q)-code, where n defines the block size, k defines the number of source data symbols encoded in the block, n-k defines the number of parity symbols added to the source data to form the block, d_h defines the Hamming distance of the code, and q defines the base of each symbol in the block. The error detection and correction capability of this block code depends solely on its Hamming 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.

A block code can detect up to d-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 floor((d-1) / 2) symbol errors in the block when the

receiver does not have *a priori* knowledge of which of the symbols in the block are in error. There are three classes of block code that may be used as the high-level code component of LHECC.

The simplest class of block codes is called "*checksum*". Checksum is a (n, n-1, 2, q) code and supports at most one parity symbol. Its Hamming distance is 2, meaning that it has the ability to detect one error and correct this error if it is an erasure. It cannot correct any symbol errors if the index within the block of the symbol in error is not known. Its most important advantage is that it places no restrictions on the symbol base q or the block size n. In order to calculate the parity symbol for a checksum code, the encoder performs a base-q addition of the symbols in the code block and uses the sum as the parity symbol.

Another class of block code is called "*maximum distance separable (MDS) codes* [62]." MDS codes are defined as any (n, k, n-k+1, q)-code. The most widely-used and well-known MDS code is the Reed-Solomon Code. MDS codes meet the Singleton bound, meaning that the code's Hamming distance is defined by $n - k + 1$. In other words, the code is capable of correcting a maximum number of erasures within each block equaling the number of parity symbols. It can also correct a maximum number of errors within each block equaling half of the number of parity symbols (rounded down). Codes that meet the Singleton Bound require the lowest possible number of overhead (parity symbols) relative to their corresponding error correcting capability. In other words, they have the highest possible amount of error correcting power given n and k. In this sense, they are provably optimal relative to code efficiency. However, MDS codes have two important restrictions that place limitations on the conditions under which these codes may

be used. First, the value of q must be either a prime number or a power of a prime number. The second restriction is that the maximum block size, or n, is defined as q+1 [32].

The third class of block code is called “almost maximum distance separable codes”, or AMDS codes [23, 24]. AMDS codes are defined as any (n, k, n-k, q)-code that has a Singleton defect of 1. This means that one additional parity symbol is required to achieve the same distance (and error correcting power) as an MDS code, or $d = n - k$. In other words, the code can correct a maximum number of erasures in each block equaling one less than the number of parity symbols, and correct a maximum number of errors in each block equaling half of one less than the number of parity symbols. This class of code also requires that q be a prime number or a power of a prime number. However, it does not have the same restriction on block size as MDS codes. The maximum size of an AMDS code is $n = p^m+1 - 1$ where $q = p^m$.

6.6. Building an LHECC Code

LHECC reduces to a binary code set whose Hamming distance is defined by the low-level code and where each code word is made up of a sequence of valid nCm symbols. In order to build an LHECC code, the interconnect designer must decide on several parameters that ultimately determine the interconnect’s overall physical width, data capacity, and error correcting power. These parameters include the MBDS channel width, number of MBDS channels, maximum number of correctable bits within an nCm symbol, and maximum number of correctable nCm symbols within the code word.

The MBDS channel width determines the nCm symbol set that is used for the partitioning operation. This width, when multiplied by the number of channels used to form the interconnect

(high-level block size), determines the overall width (in wires) of the interconnect. The target Hamming 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. In other words, the LHECC 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 floor((d_l – 1) / 2) bit errors. Likewise, the LHECC is capable of correcting up to floor((d_h – 1) / 2) nCm symbols errors, assuming these symbols are received as pure errors (valid nCm symbols) and each of these symbols contain less than or equal to floor((d_l – 1) / 2) bit errors.

Another consideration when constructing an LHECC is the value of *s* and *c* for the partitioning operation. This value becomes the base of the symbols used in the high-level code. MDS and AMDS codes require that the base of the symbols used in the code be a prime or a power or a prime number. Specific values of *s* can be targeted in the partitioning operation to suit the high-level code. In addition, choosing powers of 2 as values for *s* and *c* decrease encoder and decoder complexity as this eliminates need for base conversion hardware.

LHECC codes are formally defined as (*nCm*, *n*, *k*, d_h, *s*, *c*, d_l) codes. *nCm* defines the symbol set used to represent each of the symbols of LHECC code word. The *n*, *k*, and d_h high-level block code parameters define the block length, number of data symbols, and distance. The *s*, *c*, and d_l low-level code parameters define the number of subsets, symbols per subset, and subset distance.

6.7. Encoding LHECC Codes

Figure 6.7 shows an overview of the LHECC encoding procedure. Segments of source data are split into an s-data portion and c-data portion. The s-data portion, as represented as a sequence of base-*s* symbols, is encoded using the high-level block code. The c-data portion, as represented as a sequence of 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.

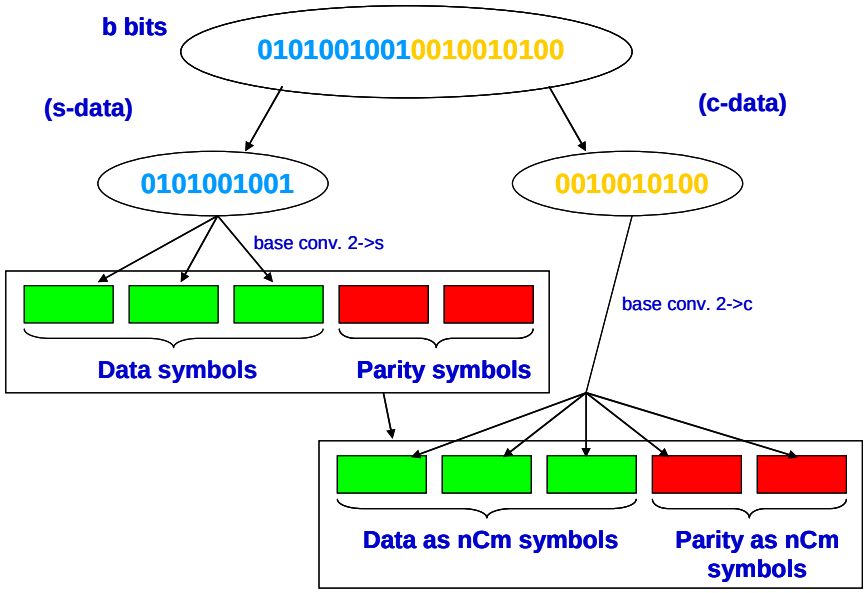


Figure 6.7: Encoding LHECC codes.

For example, assume an interconnect formed with three parallel 4c2 channels. Assume the 4c2 symbol sets are partitioned into 3 subsets, 2 symbols per subset, and subset distance of 4. The partitioning is shown in Table 6.3. Assume a high-level code consisting of a (3, 2, 2, 3) checksum. This high-level code is capable of correcting 1 erasure.

Table 6.3: Partitioning for 4c2 symbol set.

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

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 for an absolute code rate (interconnect capacity divided by interconnect width in wires) of .50. An independently mapped MBDS interconnect having three 4c2 channels would also carry 6 bits, yielding a relative code rate of 1. In other words, in this case no information overhead is required when adding LHECC support to this interconnect. Assume the transmitter wishes to encode and transmit the binary sequence "111101" across the 4c2 channels. The following actions occur at the encoder:

- The most significant 3 bits become the s-data. 111 is converted from base-2 to base-3, making the s-data become 21.
- A checksum is computed and appended to the s-data, forming the high-level code block. $2 + 1 \pmod 3 = 0$.
- 210 is the high-level code block. The least significant 3 bits of the source data becomes the c-data and is used to choose an nCm symbol from the subsets specified by the high-level code block. Using Table 6.3, 210 is combined with 101 to form the LHECC code word, yielding 1001 0101 1100.
- The code word is used as an input to three parallel 4c2 MBDS drivers.

6.8. Decoding LHECC Codes

The receiver decodes the LHECC code word by following a discrete sequence of steps. First, each nCm symbol making up the LHECC code word is "resolved" into its corresponding s-data

symbol, c-data symbol, and an error flag. The error flag indicates if the symbol is 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 contains too many errors to be decoded. There are two possible ways this can occur. First, the decoder performs a check to determine if there are more erasures in the code word (signaled by error flags) than the high-level block code can correct. An example of this occurs when the high-level code is capable of correcting two erasures but there are three invalid nCm symbols in the code word. These types of errors are considered "low-level errors." Also, the high-level code block is checked to ensure that it doesn't contain more "non-erasure" errors than can be corrected by the high-level block code. This can occur if symbol errors exist that aren't detected as invalid nCm symbols and the high-level code block constitutes a sequence of values that it is not correctable. An example of this occurs when a checksum code is used and there is a checksum failure in the high-level block code. Another example is when a high-level code is used that can correct one error and two nCm symbols contain two bit errors and the result of both are valid nCm symbols. These types of errors are considered "high-level errors."

Assuming that there are correctable reception errors, the corrected s-data values from the s-data high-level block code are computed for each nCm symbol received in error. This allows the successful extraction of the s-data (and thus subset values). These corrected s-data values along with each sampled symbol that contain bit error(s) are used to determine the original nCm symbols from the code word. This is performed by determining which nCm symbol within the

subset specified by the corrected s-data value has minimum distance relative to the received symbol. If there are multiple nCm symbols within this subset that yield equal distance to the received code word, a flag is raised indicating that the code word cannot be decoded. This situation would prevent extraction of the c-data. 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 section, 1001 0101 1100, is transmitted across the interconnect but noise affecting the channels causes the receiver to sample the following three symbols: 1101 0101 1100. 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 ? 1 0 because the most significant symbol is considered an erasure. Checksum codes have the ability to correct erasures, so combinational logic in the decoder indicates that $0 - 1 \pmod 3 = 2$.
- 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.
- Of the two symbols in subset 2, 0110 differs from the received symbol in 3 bit positions, while 1001 differs in only 1 bit position. The symbol is corrected to 1001.
- The decoder now has enough information to determine the original source data.

6.9. Minimum Distance Decoding and Miscorrection

Traditional error control codes rely on a concept called *minimum distance decoding* (also referred to as *maximum likelihood decoding*). The fundamental tenant of this type of decoding is that fewer numbers of errors are more likely to occur than greater numbers of errors. Therefore, assume block *X* is received that doesn't match any of the valid code words contained within the code set. If *X* resembles one of the code words, *A*, more closely than any other code word within the code set, the decoder will assume that *X* was originally transmitted as *A* and will perform correction based on this assumption.

Now, assume that the Hamming distance of the code set is 4, meaning that each code word differs from every other code word by at least 4 symbol positions. Assume the originally transmitted code word was *B*, but it was received containing symbol errors as *X*. If *X* was the result of 3 symbol changes from *B*, then *X* may differ in only 1 symbol position from code word *A* and will be wrongly corrected to *A*. In other words, any error that results in greater than $d/2$ symbol errors will result in a miscorrection, where *d* is the Hamming distance of the code set. Fortunately, this isn't a critical problem in traditional error control codes because the distance of the code set is chosen based on the severity of expected errors. In other words, the power of an error control code is driven by expected noise and error characteristics of the channel over which it is used.

Like other error control codes, LHECC relies on minimum distance decoding. As such, it is theoretically possible to miscorrect at both the high-level code and the low-level code. For example, a low-level code miscorrection may occur if the nCm symbol set is not partitioned with high enough Hamming distance for the numbers of bit errors that occur within an nCm symbol.

Simulation of the MBDS-based interconnects described in Chapter 7 indicate that the severity of errors that would trigger such a miscorrection do not occur until signal integrity has degraded to the point at which bit errors occur uniformly across the block, causing the decoder to judge the block as having detectable but uncorrectable errors. In other words, it is extremely unlikely that a large concentration of bit errors would occur within only one or a few symbols within a block, which is the only type of occurrence that would cause a low-level miscorrection to occur. High-level miscorrections are even less likely. This can only occur if multiple bit errors within multiple symbols resulted in every symbol in the code word being a valid nCm symbol, and the resultant sequence of s-data symbols and high-level parity symbols match a valid high-level code word.

6.10. LHECC Theory

As shown in Figure 6.8, LHECC code words are encoded from binary source data that is formed from the s-data and c-data. Assume that the Hamming distance within each subset (subset distance) is 4, meaning that any pair of symbols chosen randomly from any single subset differ in at least 4 bit positions. Also, assume that the inter-subset distance is 2, meaning that any pair of symbols chosen from different subsets differ in at least 2 bit positions. Finally, assume the distance of the high-level code is 2. Now, assume an LHECC code word has been encoded based on a random initial value of s-data and c-data. Any change in the c-data will cause a single nCm symbol within the code word to switch values within its own subset, guaranteeing that the net effect of this change will be at least 4 bit positions from the previous code word. On the other hand, any change in the s-data will induce a change in at least 2 symbols in the high-level block code (due to the high-level distance). This will change the subset designators of at least two of the code word's nCm symbols and thus force a change in these symbols across different

subsets. Even if the inter-subset change in each of these nCm symbols resulted in 2 bit positions each, the total change across the code word would be 4. This means the optimality of the LHECC code relative to code density and ECC capacity is a function of how well the symbol sets are partitioned.

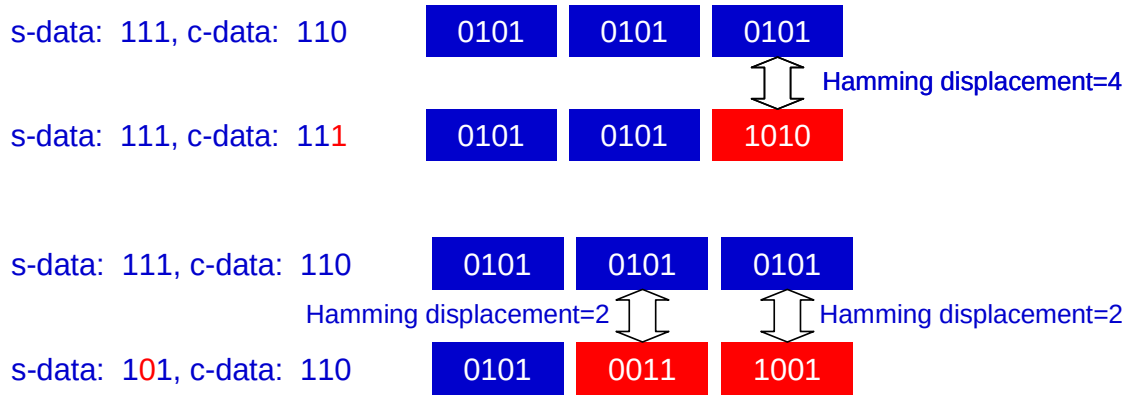


Figure 6.8: Effects of changing s-data or c-data for an LHECC encoding for a 4c2 symbol set, partitioned into 3 subsets, 2 symbols per subset, subset distance of 4, inter-subset distance of 2.

6.11. Code Rate and Overhead

There are several ways to measure information efficiency for an LHECC code. One way is to determine the information capacity lost as a result of LHECC encoding. This is computed by comparing the information capacity of an LHECC-encoded MBDS interconnect relative to that of an non-LHECC-encoded MBDS interconnect formed by an equal number and type of independently-mapped MBDS channels. This method is called *relative code rate*, and is defined as the number of bits that are carried by the LHECC-encoded MBDS interconnect divided by the number of bits carried by the non-LHECC-encoded MBDS interconnect. *Overhead* is another way to express the same measurement, and is defined as the difference in bit capacity between these two interconnects.

Another measurement is required in order to compare the efficiency of an LHECC-encoded MBDS interconnect relative to interconnects built with competing technologies. This is

accomplished using *absolute code rate*, which is defined by the bit capacity of an LHECC-encoded interconnect divided by the number of wires that form the interconnect.

The capacity of an LHECC-encoded MBDS interconnect is defined by the number of channels that form the interconnect, the number of parity symbols used by the high-level code, and the values of *s* and *c*. The high-level code carries $k * \log_2(s)$ bits, while the low-level code carries $n * \log_2(c)$ bits. The total capacity of the interconnect is the sum of each of these values rounded down, or $\text{floor}(k*\log_2(s)) + \text{floor}(n*\log_2(c))$. The capacity of an equivalent non-LHECC-encoded MBDS interconnect is $n*\text{floor}(\log_2(\textit{symbol set size}))$. The relative and absolute code rates are therefore defined as:

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

Overhead in an LHECC code word is defined as the effective number of bits carried by the code word that are reserved for parity information. However, the only components of an LHECC that are strictly used for parity information 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. This way, the only components of the code word that are used only for parity information are the parity values in the high-level code block. Also, relative to the high-level code block, fewer parity symbols (and less Hamming distance) are required to achieve error correction capability as compared to traditional block codes. 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 pure errors. This low overhead requirement makes LHECC lightweight. Overhead for LHECC 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 computed 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. Table 6.4 shows several ways nCm symbol sets can be partitioned and the resultant overhead required for each parity symbol used in the high-level code. In this table, overhead is expressed in terms of bits as well as amount of data relative to the information capacity of each nCm symbol.

Table 6.4: nCm symbol set partitionings.

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

6.12. LHECC Examples

This section describes encoding and decoding examples of multiple-symbol correcting LHECC configurations. These examples have more sophisticated high-level codes and more error correcting power than in the previous example. Each example outlines the LHECC encoding and decoding process and how errors are detected and corrected.

6.12.1. (4c2,4,2,3,3,2,4) LHECC, Correcting Two Erasures

This example is based on an interconnect built from four parallel 4c2-channels and illustrates a code that is capable of correcting two nCm symbols. The 4c2 symbol sets are based on the same partitioning as in the previous example, shown again in Table 6.5.

Table 6.5: Partitioning of a 4c2 symbol set.

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

The high-level block code is a (4, 2, 3, 3)-MDS code. It has the capability to correct 2 erasures or 1 error. Assume the following systematic generator matrix and associated parity-check matrix are used for the high-level code:

$$G = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \end{bmatrix} \qquad H = \begin{bmatrix} 2 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{bmatrix}$$

This example code encodes 3 bits into its s-data and 4 bits into its c-data, yielding an interconnect capacity of 7 bits using 16 wires for an absolute code rate of .44, a relative code rate of .88, and 1 bit of overhead.

Assume the following data is to be transmitted: 1001010. The following actions occur at the encoder:

- The most significant 3 bits, 100, become the s-data and are converted to base-3, becoming 11.
- The s-data is encoded into a code block by effectively multiplying [1 1] by the generator matrix. A table lookup operation, mapping data symbols to corresponding parity

symbols, is used to determine the parity symbols. The high-level code block becomes [1 1 2 0].

- The least significant 4 bits become the c-data. 1010, is encoded by choosing nCm symbols corresponding to the subsets specified by the s-data block code, yielding 1010 0101 1001 0011.

Noise affecting the channels causes the receiver to sample the following four symbols: 1110 0101 1001 0001.

The following actions occur at the decoder:

- The most significant and least significant symbols are not valid 4c2 symbols. These symbols are treated as erasures for decoding the block code.
- The code word is resolved to a high-level code block of ? 1 2 ?. The erasures can be corrected via table lookup based on the observation that the code space of the block code contains only one possible block matching the correctly received values. The high-level code space is shown in Figure 6.8. The block code is corrected to become [1 1 2 0].
- Minimum distance decoding is performed on the two symbols that were received in error, by comparing the received symbols versus the symbols in each subset. This yields the corrected symbol sequence and the original source data.

$$m \bullet G = C$$

$$\begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 2 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 2 & 0 \\ 2 & 1 \\ 2 & 2 \end{bmatrix} \bullet \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 2 \\ 0 & 2 & 2 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 2 & 0 \\ 1 & 2 & 0 & 2 \\ 2 & 0 & 2 & 2 \\ 2 & 1 & 0 & 1 \\ 2 & 2 & 1 & 0 \end{bmatrix}$$

Figure 6.9: Code space of a (4,2,3,3) code, derived from multiplying the message space by the generator matrix.

6.12.2. (4c2,4,2,3,3,2,4) LHECC, Correcting One Error

Assume the same LHECC as in the previous example. As in the previous case, assume the following code word is transmitted:

1010 0101 1001 0011,

with the receiver sampling:

0110 0101 1001 0011.

In this case, 2 bit errors occurred within the most significant symbol yielding a valid 4c2 symbol. The original high-level code block is [1 1 2 0], but the received code word is resolved to the high-level code block [2 1 2 0]. In this case, all received symbols are valid nCm symbols and no erasures have been detected. From the standpoint of the high-level code, a correctable error has occurred. In order to cope with this error, the decoder logically performs a syndrome computation using a table lookup. Conceptually, the received block code is multiplied by the transpose of the parity-check matrix (H^T), yielding a syndrome of [2 2]. A non-zero syndrome

indicates that error exists in the high-level code block. This syndrome value corresponds to an error vector of [1 0 0 0] because this is the only possible error vector, assuming one error, that when multiplied by the transpose of the parity-check matrix yields a matching syndrome value. The error vector is logically subtracted from the received code block via table lookup, yielding a corrected code block of [1 1 2 0].

$$e \bullet H^T = S$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 \end{bmatrix} \bullet \begin{bmatrix} 2 & 2 \\ 2 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 2 & 2 \\ 1 & 1 \\ 2 & 1 \\ 1 & 2 \\ 1 & 0 \\ 2 & 0 \\ 0 & 1 \\ 0 & 2 \end{bmatrix}$$

Figure 6.10: All possible error vectors multiplied by transpose of parity-check matrix, yielding syndrome space.

The next step in the error correcting process is to perform a minimum distance calculation between the received symbol, 0110, and the symbols in subset 1, being 0101 and 1010. However, in this case, the received symbol differs in two bit positions from both symbols in subset 1. The result is an uncorrectable error, due to the fact that the original partitioning operation targeted a subset Hamming distance of 4. This is not enough to correct a 2 bit error occurring within a single symbol. In this case, the s-data may be recovered while the c-data cannot.

6.12.3. (6c3,4,2,3,9,2,6) LHECC, Correcting One Error

In order to correct double-bit errors occurring within a single symbol, an nCm symbol set must be chosen that may be partitioned with a Hamming distance of 6. In this example, a 6c3 symbol set is partitioned into 9 subsets with 2 symbols per subset, achieving a subset Hamming distance of 6. While it is possible to partition this symbol set into 10 subsets of 2 symbols per subset, the high-level MDS code requires that the symbol base used be a prime number or power of a prime number. For this reason, only 9 subsets are used. The partitioning is shown in Table 6.5.

Table 6.6: Partitioning for 6c3 symbol set into 9 subsets with 2 symbols per subset and target Hamming distance 6.

	c	
s	**0**	**1**
0	000111	111000
1	001011	110100
2	001101	110010
3	001110	110001
4	010011	101100
5	010101	101010
6	010110	101001
7	011001	100110
8	011010	100101

Assume an LHECC based on this partitioning as the low-level code and a (4, 2, 3, 9) MDS block code as its high-level code. The generator and parity-check matrices for this high-level code are shown below. In this case, 6 bits are encoded into the s-data and 4 bits are encoded into its c-data. This yields an interconnect capacity of 10 bits over 24 wires, for an absolute code rate .42, a relative overhead of .63 (based on 16 bits of non-LHECC capacity), and 6 total bits of overhead.

$$G = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \end{bmatrix} \qquad H = \begin{bmatrix} 8 & 8 & 1 & 0 \\ 8 & 7 & 0 & 1 \end{bmatrix}$$

This code is capable of correcting several classes of errors. It can correct a single symbol containing a 2-bit error when the symbol is a valid 6c3 symbol or up to two symbols containing a 1 or 2-bit error if the resultant symbols are not valid 6c3 symbols. The high-level block code is capable of correcting 2 erasures with up to 2 bit errors in each symbol or 1 error with up to 2 bit errors in the symbol. Assume the following code is transmitted: 010110 110010 100101 001011, containing a high-level code block of [6 2 8 1]. Assume 2 bit errors occur within a single symbol, yielding a code word of 010110 110010 **101001** 001011. The received code word contains no invalid symbols and is resolved to a high-level block code of [6 2 **6** 1]. The syndrome of this block code is [7 0], matching the syndrome value of a [0 0 7 0] error vector. The error vector is used to determine the subset number of the symbol in error. The symbol containing 2 bit errors, when compared to the symbols from its originating subset, will differ in 4 bit positions from one and 2 bit positions from the other. The symbol in error, 101001 is a member of subset 8, which contains 011010 and 100101. The symbol is corrected to 100101.

6.13. LHECC Architectures

The LHECC encoder and decoder are responsible for converting source data between binary and LHECC representation as well as for correction and detection of errors. This section describes an example implementation of an LHECC encoder and decoder architecture. In order to fulfill the goals set forth in Chapter 5, it must be viable to integrate these components into the I/O interfaces of system-level interconnect. To meet this goal, these components must be extremely compact and capable of operating at the same operating frequency as the interconnect. LHECC has several properties that make this possible. First, due to the low relative overhead of LHECC encoding, the encoders/decoders may be designed for codes with small block lengths of 3 - 4 symbols. Small block sizes serve to reduce overall decoding complexity, because the

fundamental, core operations within the encoder and decoder can be based on combinational lookup logic rather than arithmetic operations. Second, the partitioning operation effectively reduces block code decoding complexity because it ensures that the symbol base of the symbols of the high-level code is less the base of the nCm symbols themselves. Symbol base is a major factor in the complexity of the high-level encoder and decoder. Also, the hierarchical nature of the encoding and decoding process requires a discrete sequence of steps that naturally lends itself to a pipelined implementation.

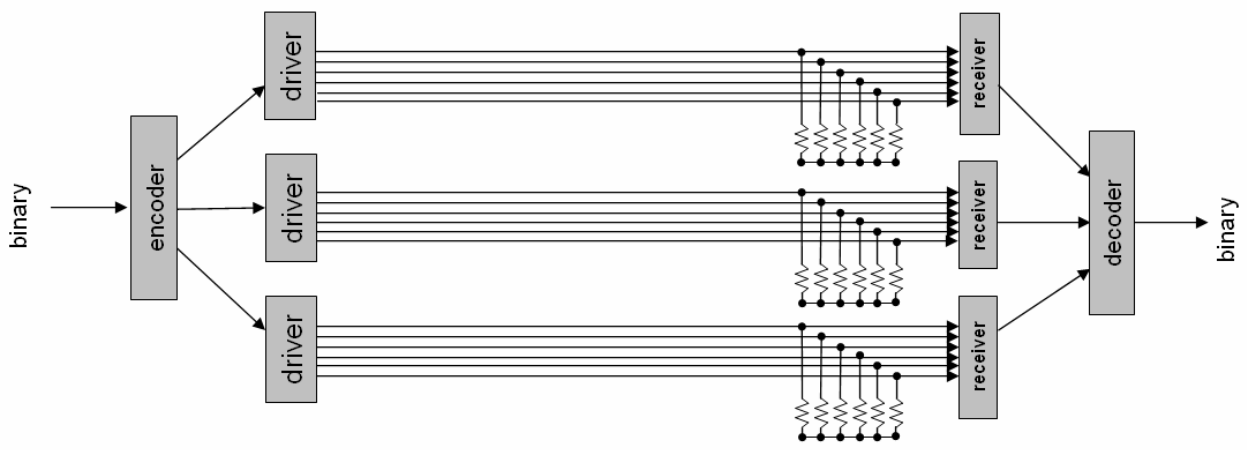


Figure 6.11: Example organization of an LHECC-encoded MBDS interconnect with 3 6c3 channels. Each channel has an independent MBDS drivers, receiver, and termination network.

In order to ensure that these encoders and decoders are capable of operating at core frequencies, they are pipelined where each pipeline stage consists of combinational logic with at most 8 gate delays in their critical paths. The encoders and decoders are designed such that source data is completely encoded and completely decoded every clock cycle, ensuring a steady, uninterrupted stream of data through the interconnect.

The example encoder and decoder that is illustrated in this chapter is based on the three-channel LHECC interconnect shown in Figure 6.11. As shown in this figure, the encoder drives three

independent 6c3 MBDS drivers. Likewise, the decoder receives data from three independent 6c3 MBDS receivers. The LHECC code for this interconnect consists of three 6c3 symbols with the 6c3 symbol set partitioned into 4 subsets of 4 symbols per subset and using a high-level block code consisting of a (3, 2, 2, 4)-checksum. This code has a capacity of 10 bits and requires 18 wires, yielding a relative code rate of .83, an absolute code rate of .56 and overhead of 2 bits. This code can correct one bit error occurring within one symbol.

The example encoders and decoders described in this chapter are built using a standard cell library consisting of the following components: 1-bit latch, inverter, 2 and 3-input NAND gate, and 2 and 3-input NOR gate. Each combinational block in the designs are synthesized VHDL.

6.13.1. Encoder Architecture

An example encoder is shown in Figure 6.8. This encoder is divided into 2 pipeline stages and requires 208 gates and 30 latches. For this code, the base of the s-data and the c-data are powers of 2, so no base conversion is required. If base conversion was required, an additional stage would be inserted before the first pipeline stage shown here. For example, a base-2 to base-3 converter can be built with 19 gates and a critical path gate delay of 4.

The “high-level code” stage of the encoder is where the checksum parity symbol is computed from the two 2-bit (base-4) s-data values. This component requires 19 gates and 5 gate delays. More sophisticated codes can also be encoded in one stage. For example, a (4, 2, 3, 3) MDS code, which requires generation of 2 base-3 parity symbols, can be built with 37 gates and 5 gate delays. The complexity of this stage grows with the size of the code space of the high-level

code. The size of the code space is equal to the size of the message space, or every possible value of the "source data" portion of the high-level code block. For a message of width k formed with symbols of base q, the size of the code set is q^k. Therefore, the complexity of this stage grows quadratically with symbol base and exponentially with high-level code block length. This stage requires less complexity relative to a traditional ECC code used over these MBDS channels because the partitioning operation guarantees that the symbol base of the high-level block code is less than the base of the nCm symbols transmitted over the interconnect. This represents another way that LHECC reduces the overall encoding complexity for ECC over system-level interconnect.

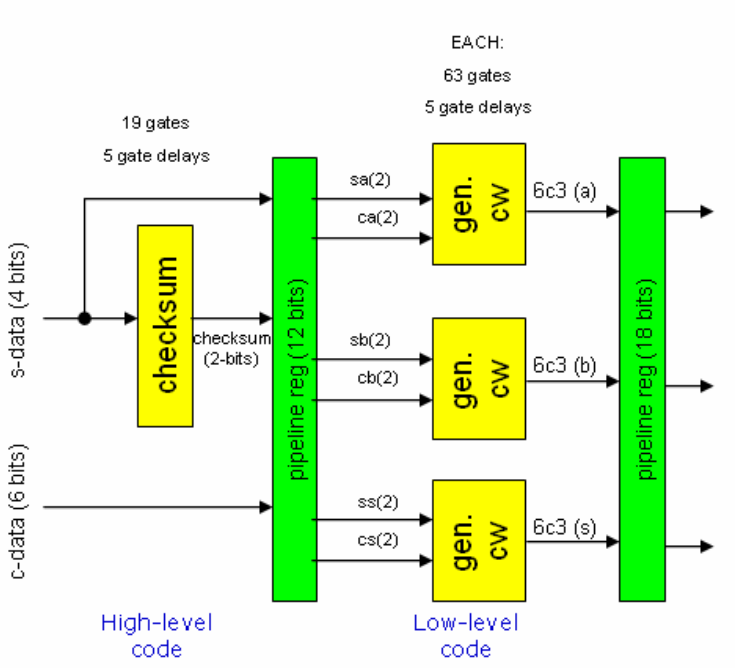


Figure 6.12: Example encoder architecture.

The "low-level" stage of the encoder is responsible for generating inputs to the MBDS drivers based on the s-data and c-data values for each symbol in the code word. In this case, each of these components requires 63 gates with 5 gate delays each. The complexity of this stage grows

linearly with the size of nCm symbols set. The final pipeline register is used to ensure the inputs to each of the MBDS drivers remains stable for a full clock cycle.

6.13.2. Decoder Architecture

An example decoder is shown in Figure 6.13. This decoder is divided into 5 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 and c-data values. If any of the sampled symbols is not valid, an error flag is raised by the resolver. Each of these resolvers requires 130 gates and 7 gate delays. The complexity of these components grows linearly with the size of the nCm symbol set.

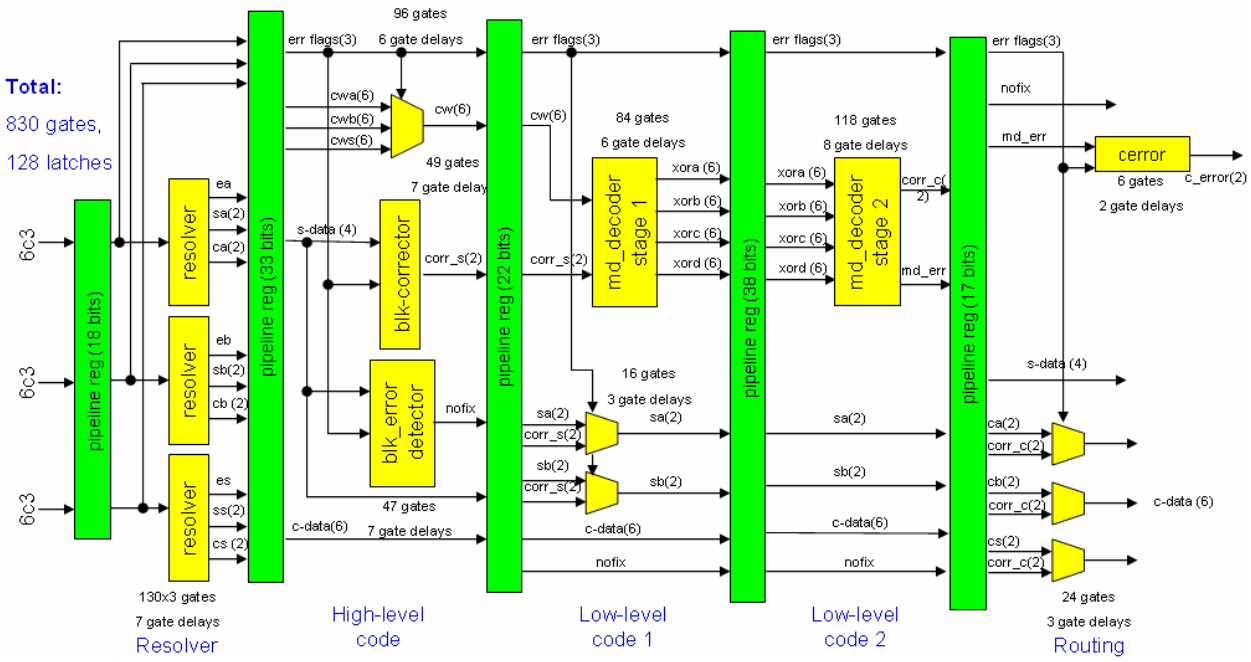


Figure 6.13: Example decoder architecture.

The “high-level code” stage is responsible for managing the high-level code. This involves performing preliminary checks for non-decodable code words and high-level block code correction. The “block error detector” determines if there are too many errors in the code word to correct. This occurs if there are too many erasures (error flags) or if there are no erasures but the high-level checksum is invalid. This component requires 47 gates and 7 gate delays. The “block corrector” component is responsible for determining the subset value(s) of nCm symbols that are received in error. For this code, the corrector will determine the subset value for any single nCm symbol that contains bit errors that result in an invalid nCm symbol. For this code, this component requires 49 gates and 7 gate delays. The complexity of this stage grows with the use of wider block codes relative to block length, block code distance, and block code symbol base (number of subsets). For example, for a (4,2,3,3) block code (allowing the correction of 2 erasures or one error), the block code error detector requires 106 gates and 7 gate delays, while the block code error corrector requires 364 gates and 7 gate delays. As in the encoder, this stage requires less complexity relative to a traditional ECC code, since the partitioning operation guarantees that the symbol base of the high-level block code is less than the base of the nCm symbols transmitted over the interconnect. As with the high-level code stage in the encoder, the complexity of this stage grows with the size of the code space of the high-level code. The size of the code space is equal to the size of the message space, or every possible value of the “source data” portion of the high-level code block. For a message of width k formed with symbols of base q, the size of the code set is q^k. Therefore, the complexity of this stage grows quadratically with symbol base and exponentially with high-level code block length.

"Low-level code 1" and "low level code 2" are two stages devoted to 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 2 errors are present in the symbol, a flag is raised indicating that the errors cannot be corrected due to the target Hamming distance used in the partitioning operation. The complexity of this operation is a function of the width of the nCm symbols used in the interconnect. Together, these stages require 202 gates and require 6 gate delays in the first stage and 8 gate delays in the second stage. Finally, the "routing" stage contains logic for determining if the code word is not decodable and additional routing logic. The complexity of this stage grows linearly with the size of the lookup table for this operation, which is the product of the width of the nCm symbol set and the number of subsets.

6.13.3. Additional Encoder/Decoder Architectures

Table 6.7 shows statistics for four different LHECC encoder/decoder architectures. These architectures demonstrate that LHECC encoders and decoders require a relatively small amount of logic and can be operated at system core speeds.

Table 6.7: 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/1 sym	100	28	3	328	66	5	7
4x4c2	16	7	1 bit/2 sym	144	36	3	804	124	6	8
3x6c3	18	10	1 bit/1 sym	208	30	2	830	128	6	8
3x8c4	24	15	1 bit/1 sym	424	44	2	1246	234	6	8

The primary challenge when developing encoders and decoders with increasing interconnect widths and error correcting power involves designing each stage with a fixed critical-path gate delay latency. Wider and more sophisticated codes require increasing latencies per stage. This is particularly true in the resolver, block detector, block corrector, and minimum distance decoder components of the decoder architecture. In some cases, these components must be split into multiple stages themselves. This adds to the total latency of the decoder.

7. EXPERIMENTAL DESIGN

As described in Chapter 5, the LHECC technique must be verified in order to measure its effectiveness in increasing signal integrity for system-level interconnect. This chapter describes the experimental method used to perform this verification and determine if the benefits of LHECC outweigh the costs.

The first step of this verification is to construct realistic models of MBDS-based interconnects in order to capture error behavior over a range of transmission rates and noise characteristics. The second step is to simulate these models by sending a random sequence of data across the interconnect. For each simulation, results are gathered by inspecting the interconnect outputs relative to the inputs. These results determine how well LHECC performs in correcting the types of errors that are introduced within these channels. In this context, signal errors occur as incorrectly sampled bits at the outputs of the differential receivers. Several parameters are set for the interconnect in order to construct accurate models for system-level interconnect and its corresponding noise sources. Examples of such parameters include the type of system-level interconnect, driver/receiver circuit structure, and chip fabrication technology.

7.1. Interconnect Model

Printed circuit board (PCB)-based chip-to-chip interconnects are chosen as a model for system-level interconnect. This model is constructed as a set of parallel and independent PCB-based MBDS channel models. As shown in Figure 7.1, each MBDS channel of width *n* consists of a

model for a transmitting chip, receiving chip, PCB transmission lines, and a purely resistive termination network. The transmitting chip is formed by a driver circuit model and its associated group of package transmission line models. Likewise, the receiving chip is formed by a receiver circuit model and its associated group of package transmission line models. As shown in Figure 7.1, this channel model is time-domain simulated by using process-defined CMOS voltage levels/thresholds to inject valid nCm symbols into the driver and to read out the corresponding outputs from the receiver after the latency of the channel.

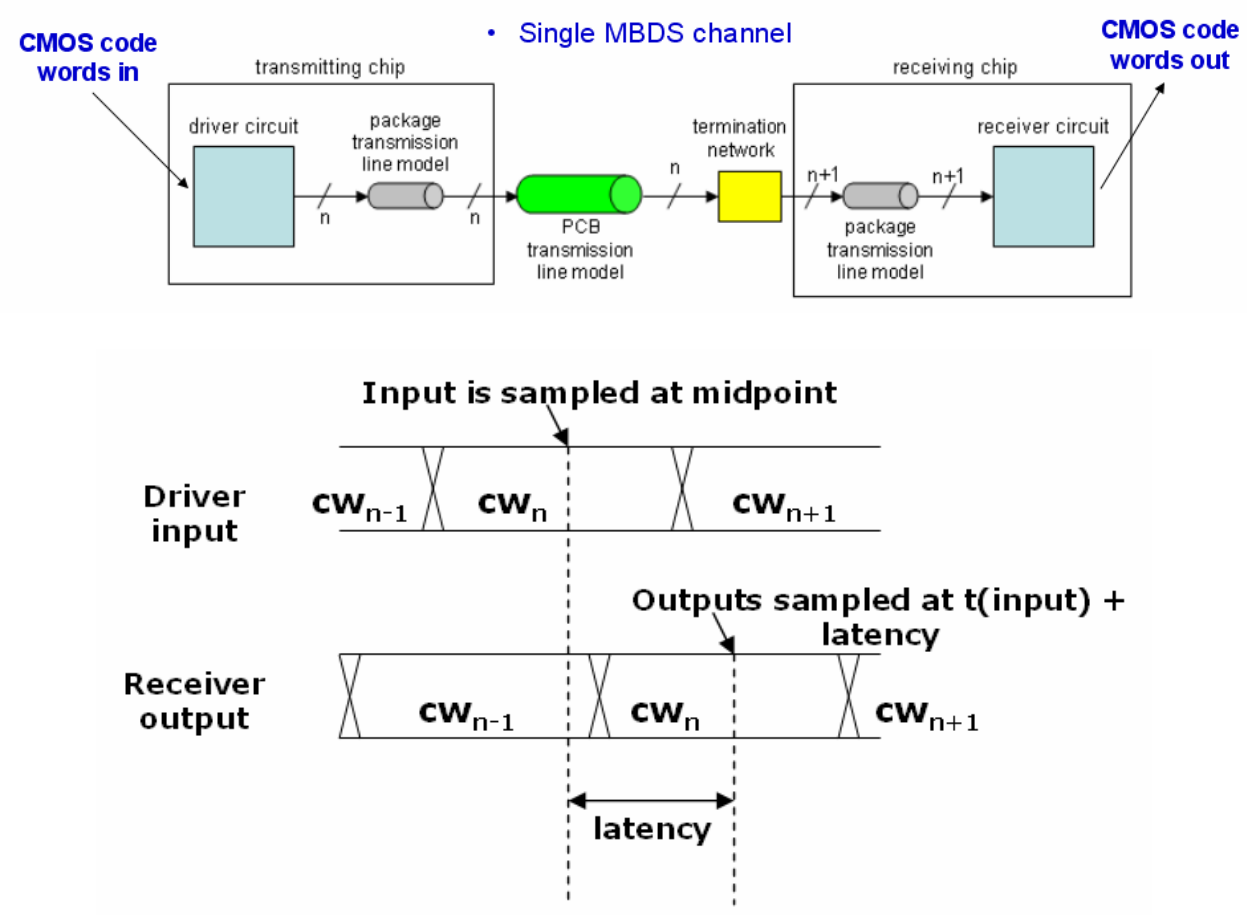


Figure 7.1: PCB-based MBDS channel model.

7.2. Driver and Receiver Circuit Models

The MBDS channel model includes driver and receiver circuits. These circuit designs are modeled using MOSFET transistor models from the Taiwan Semiconductor Manufacturing Corporation (TSMC) .18 um process design kit (PDK). These models are based on widely-used industry-standard devices that correspond to a state-of-the-art CMOS fabrication process.

Figure 7.2 shows an example driver circuit. The driver circuits are "current-steering" designs that are scalable to support arbitrary channel widths. The drivers contain one P-type and one N-type biasing transistor for each driver output that sources current that corresponds to half the channel width. The drivers contain one current-steering "leg" for each output wire forming the width of the channel. The transistors in each driver are sized and biased such that each driver output provides a mean absolute value current of 7.5 mA. This value was chosen as a fixed parameter in order to evaluate each driver width on equal ground.

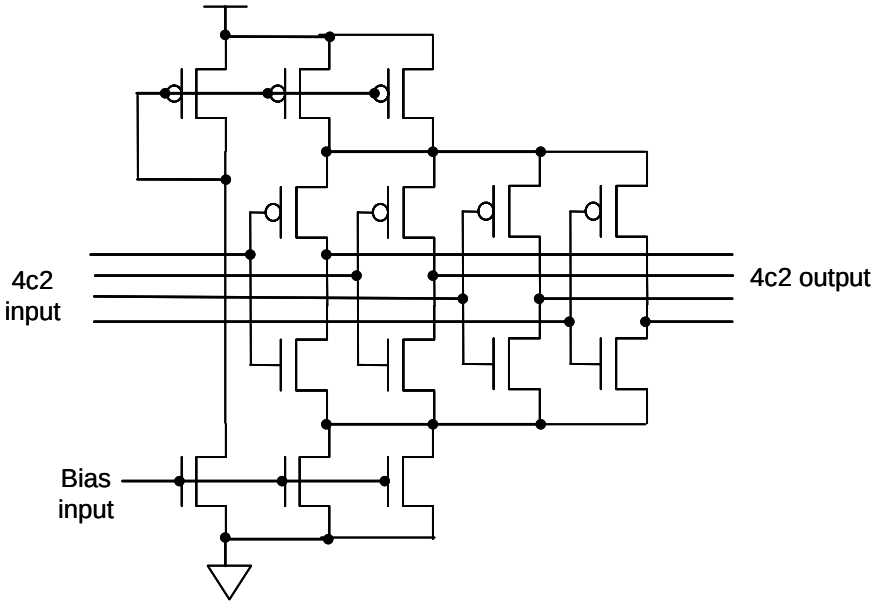


Figure 7.2: Example 4c2 MBDS driver.

The receiver circuit is shown in Figure 7.3. It is based on a low voltage differential (LVDS) receiver design from the literature [2]. This circuit is a self-biasing 2-stage differential receiver. 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.

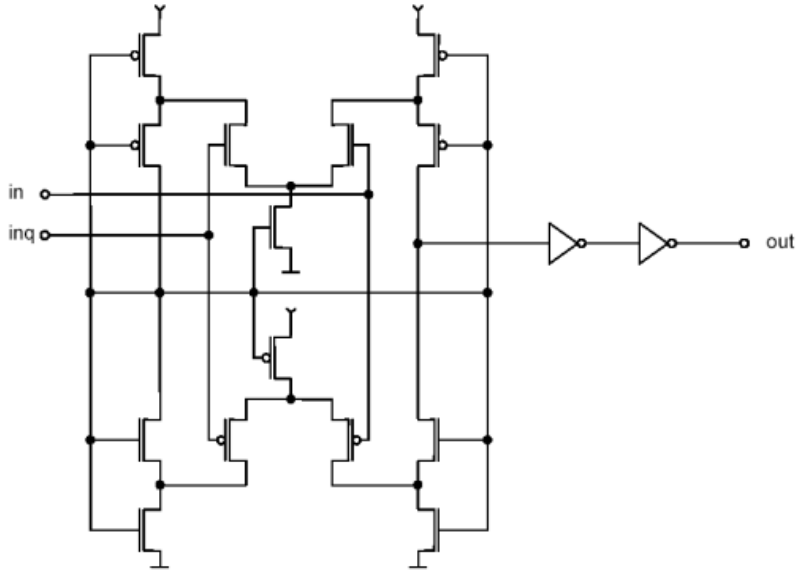


Figure 7.3: Differential receiver design [2].

7.3. Package Model

Figure 7.4 shows the chip package model. High-frequency electrical signaling from the chip, though the package, and to the PCB is subject to resistive, inductive, and capacitive (RLC) effects. For this reason, electrical connections through the package are modeled as transmission line structures.

In order to model worst-case package effects, each electrical connection from the driver/receiver circuits to the PCB is modeled as a wirebond connection. This connection consists of a model

for a wire bond connecting each chip I/O pad to a corresponding package I/O pad with models for solder bumps at both ends. The model for this connection is generated by a package modeling tool from Chatoyant [81]. The model parameters are generated using sample geometries and spacings of the chip, package, and wire bond features.

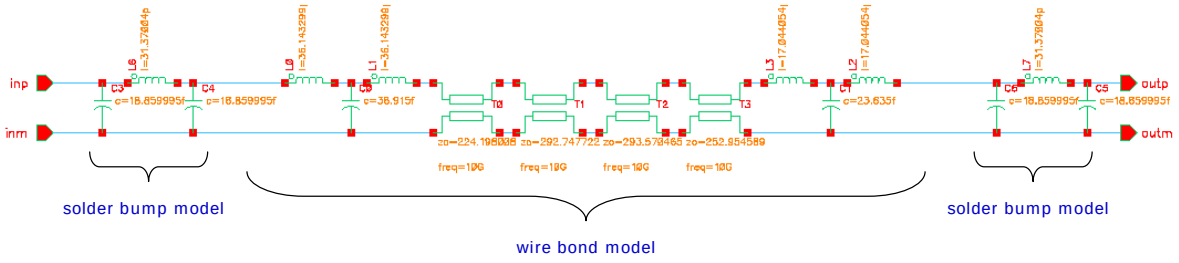


Figure 7.4: Package transmission line model for one electrical connection through the package.

7.4. PCB Trace Model

The channel model includes transmission line models for 7.5” PCB traces. PCB transmission lines exhibit frequency-dependent signal attenuation and crosstalk among near-by traces. The PCB transmission line structures used for the channel models are shown in Figure 7.5. The traces are spaced and sized to match impedance to the termination network. 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) [87]. 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.

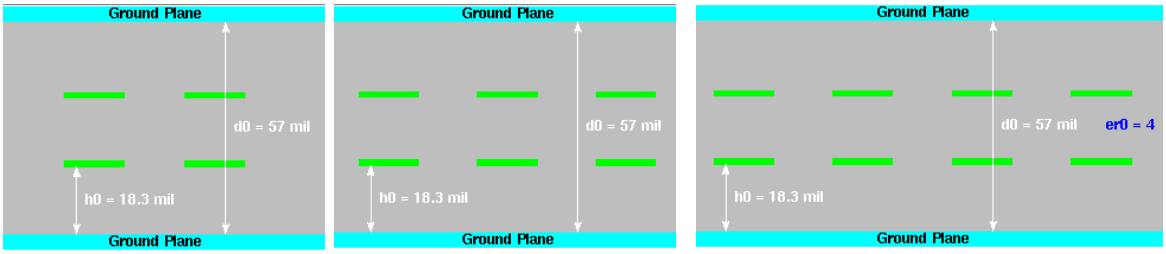


Figure 7.5: PCB trace geometries for 4c2, 6c3, and 8c4 channels.

7.5. Noise Modeling

In traditional bit-serial channels where a signal is carried on a single wire, radio frequency, or optical fiber, noise is often modeled as a random Gaussian source added to the signal on the transmission medium [37]. Using this model, signal-to-noise ratio may be used as a metric to measure noise that affects the channel. This ratio is computed as the ratio between signal power and noise power. Unfortunately, MBDS channels have more complicated noise sources due to the fact that MBDS channels are formed from multiple parallel transmission lines that are coupled to one another in various ways. Also, one of the advantages of these channels is that Gaussian noise sources on the transmission lines are filtered out at the receiver.

In addition, the channel model described in this work includes 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 and they are susceptible to additional noise sources that originate from circuitry located elsewhere on a shared chip. In addition, these circuits have their own limitations for transmitting and interpreting the signal carried by the wires that form the interconnect. The channel model described in this chapter includes several noise sources that have effects that cannot be described by a Gaussian noise source.

The channel model 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 receiver's job is to periodically sample and discriminate this signal, determining if the current signal value represents a one-bit or zero-bit. The output of the receiver is a conditioned and amplified CMOS voltage that is suitable for on-chip transmission and use. In this sense, the receiver acts as an analog-to-digital converter and amplifier. However, during sampling, if the receiver's input signal is too badly degraded, the receiver will incorrectly discriminate this signal and cause 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 and therefore capture the signal behavior over a wide range of binary sequences. 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 Tables 7.1 and 7.2. Table 7.1 shows eye diagrams of the receiver input signal and Table 7.2 shows the corresponding receiver output signal. These eye diagrams demonstrate how increasing signal degradation within 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.

Table 7.1: Eye diagrams of one receiver circuit input as symbol transmission rate is scaled along the following scale: 1.8 GHz, 2.2 GHz, 2.6 GHz, 3.0 GHz, 3.4 GHz, 3.8 GHz, 4.2 GHz, 4.6 GHz.

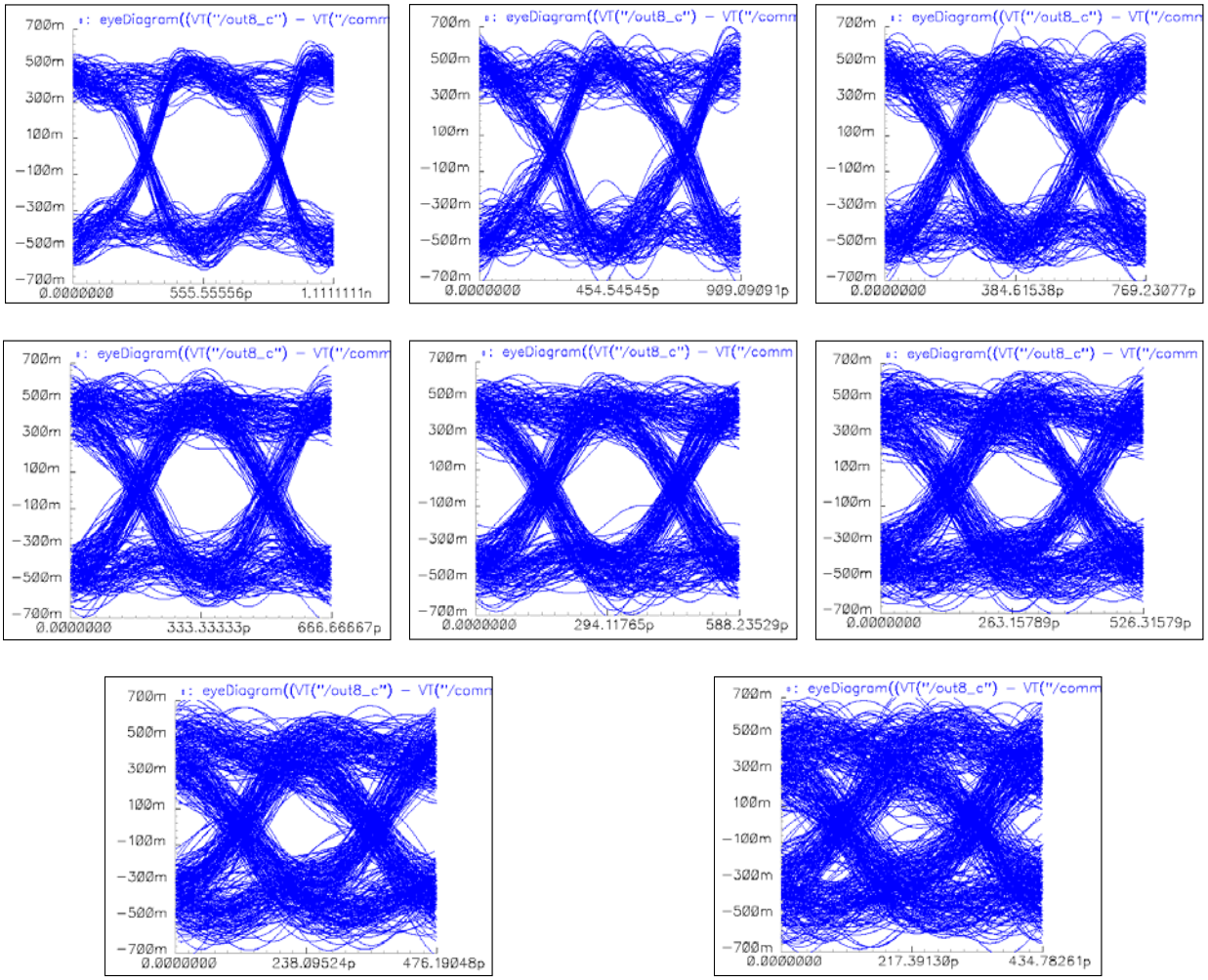
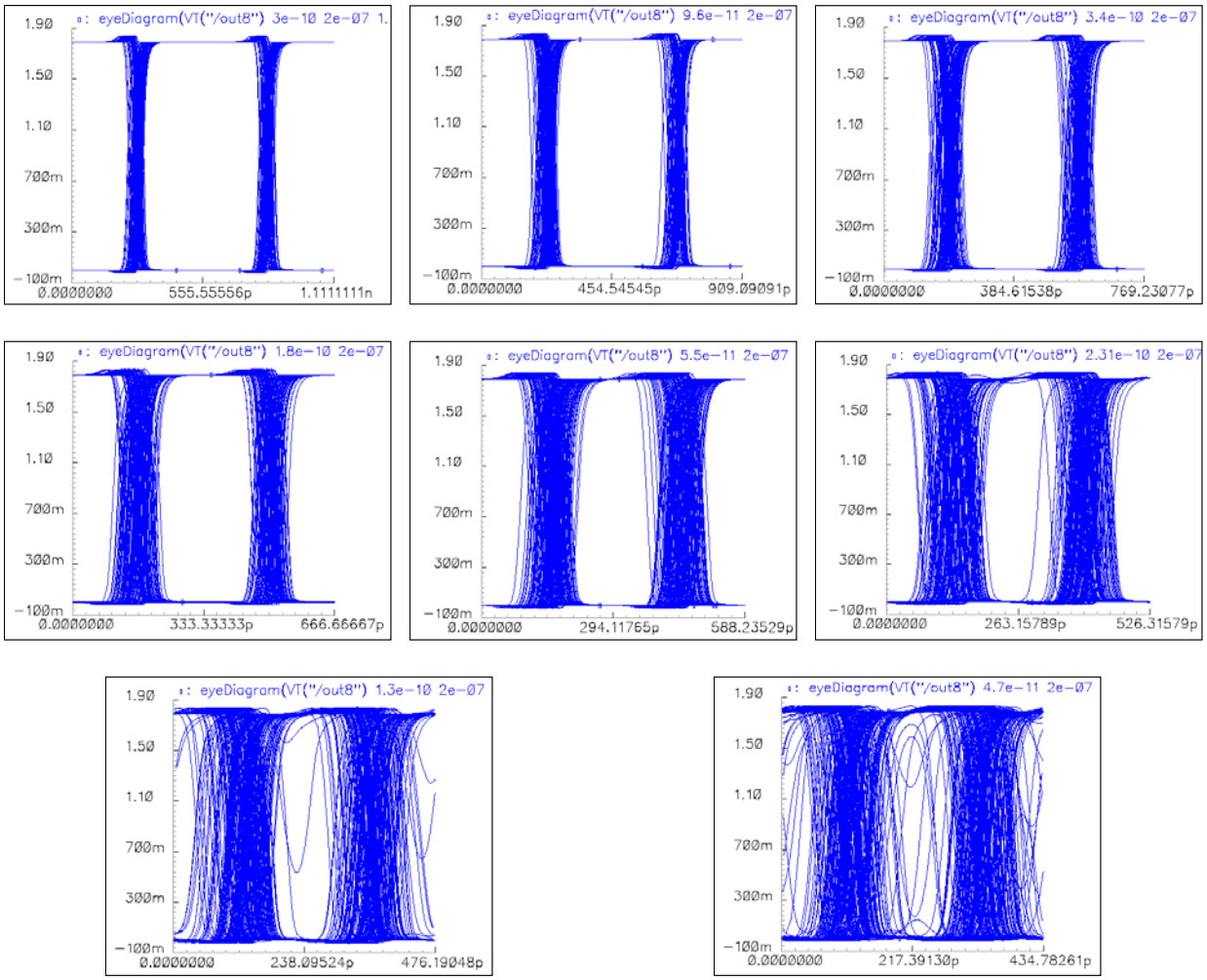


Table 7.2: Eye diagrams of one receiver output as symbol transmission rate is scaled along the following scale: 1.8 GHz, 2.2 GHz, 2.6 GHz, 3.0 GHz, 3.4 GHz, 3.8 GHz, 4.2 GHz, 4.6 GHz.



7.5.1. Supply Noise

MBDS/LHECC technology is intended for use as a system-level interface to existing large-scale digital systems, such as for interfaces for microprocessor system busses and network-on-chip switch elements. In such systems, digital logic components switch between 0 and 1 states and generate switching noise on the chip's power supply due to dynamic current transients through complimentary transistor structures. In order to illustrate this problem, Intel has measured the power supply noise on a busy Pentium 4 processor operating with a 1.5 V supply voltage [82].

This measurement was performed by probing the voltage of Vdd and ground pin pairs at two processor package locations. The voltage reading indicated a Gaussian signal centered at 1.5 V with a 17-20 mV standard deviation. This measurement is shown as an instrument screen capture in Figure 7.6.

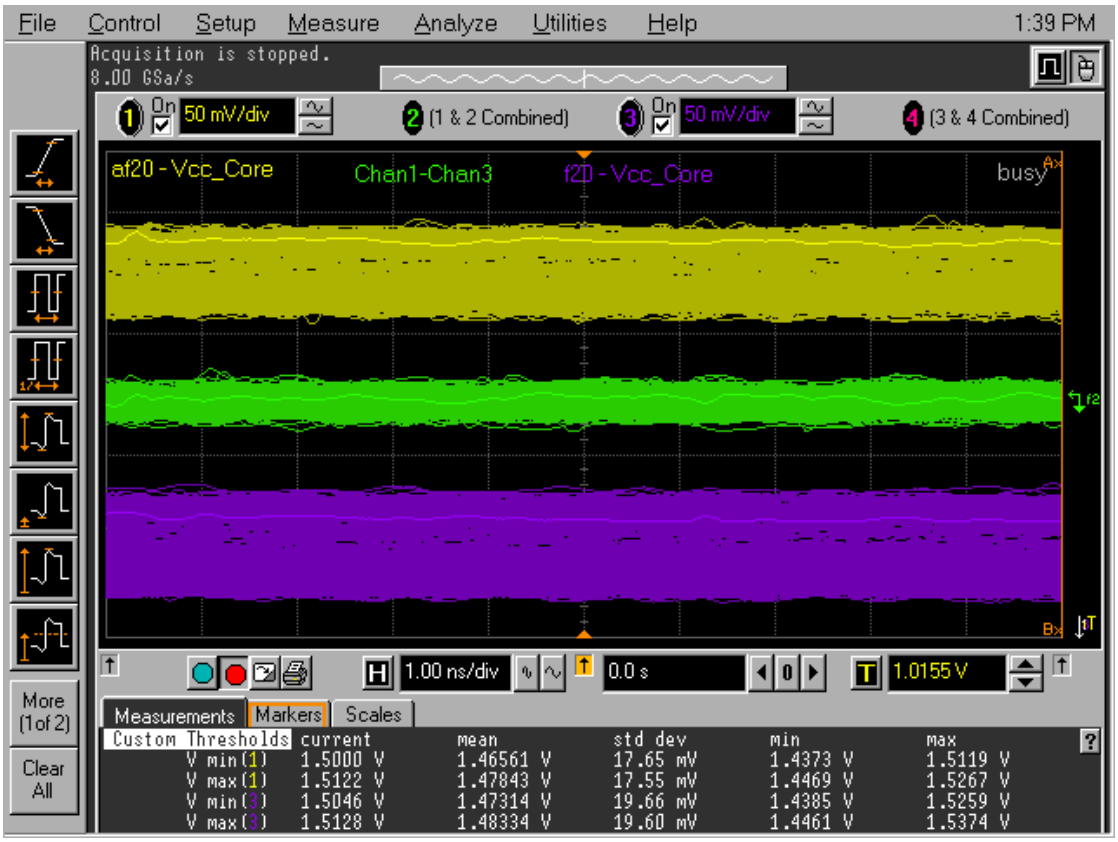


Figure 7.6: Supply measurement of a busy Pentium 4 [82].

Supply noise is an increasingly important consideration for large-scale, high-speed digital systems. Such noise would certainly affect MBDS interfaces that are used in such systems. Fluctuation in supply voltage reduces signal margins for drivers and receivers. The LHECC technique potentially 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's measurements provide statistical amplitude characteristics for supply noise from a typical microprocessor. It is clear that this type of noise is generated from transistors switching at core frequency and would thus exhibit frequency characteristics centered on the core operating frequency of the digital system. When modeling this type of noise, it is assumed that the MBDS channels, acting as a digital system interface, are operating at the same frequency as the core logic.

To generate this noise signal, white Gaussian noise is generated for 20 us 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 absolute per-wire transmission frequency of the channel that is being simulated. For example, it is assumed that a 2 GHz microprocessor would have a signaling speed of 2 Gbps. The standard deviation of the original Gaussian noise is set such that the standard deviation of the filtered signal is approximately 40 mV (to compensate for the 1.8 V supply voltage of the fabrication models and exaggerated to provide a clear picture of the effect of this noise). The noise signal is shown in time and frequency domains in Figure 7.7.

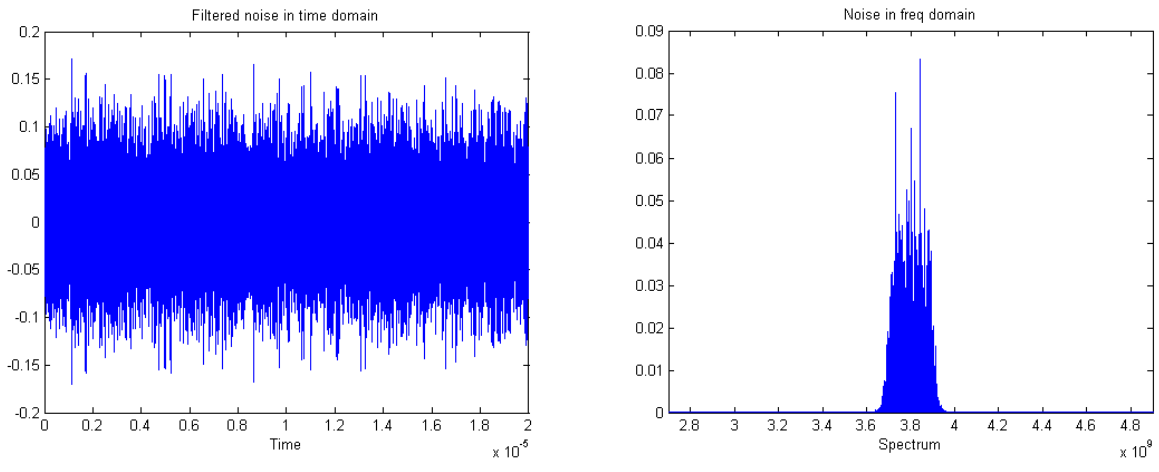


Figure 7.7: Generated supply noise in time (left), and frequency (right) domain.

7.5.2. 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. The fringe capacitance model for an 8c4 driver is shown as a schematic in Figure 7.8. Interconnect formed by parallel wires is laid out on chip as a group of adjacent and closely spaced traces. Likewise, the corresponding package I/O pads for these wires are also adjacent and closely spaced. The proximities of these features add additional crosstalk to signals within the channel. This crosstalk is modeled by parasitic capacitance between adjacent signals within each channel. For the channel model, the value of these capacitors is 500 fF, verified by the Cadence Diva extractor as reasonable fringe capacitance values.

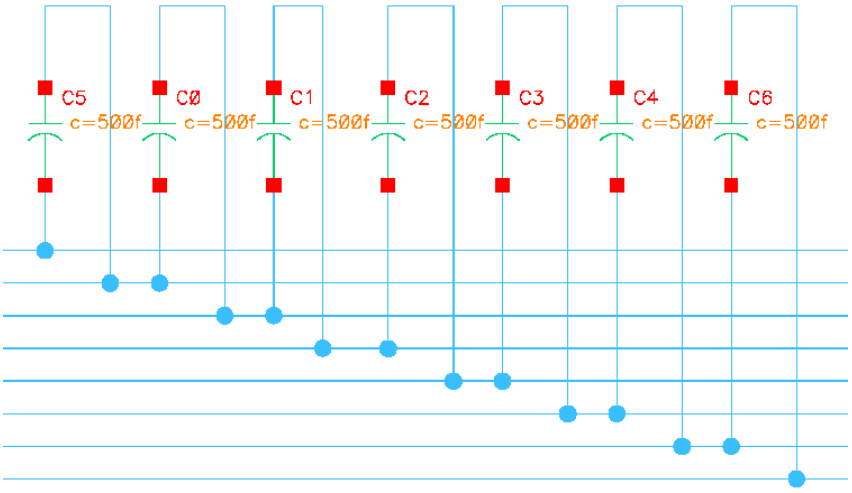


Figure 7.8: Fringe capacitance in adjacent driver outputs.

7.6. Experimental Interconnects

Chapter 8 presents results obtained through simulation of 5 different interconnect configurations. Each of these interconnects is built using the MBDS channel and noise models described earlier in this chapter. In addition, each interconnect is associated with an LHECC having various levels of overhead and error correcting power. The simulations are designed to measure the relative amount of noise immunity gained through the addition of LHECC to raw MBDS-based interconnects. This is achieved by comparing the error ratio performances of each interconnect with and without LHECC support. The interconnects are simulated with Cadence's Spectre analog simulation tool [87]. The results are collected using custom written tools that stimulate the interconnects with pseudorandom source data and count the number of signal errors by inspecting each channel's output relative to its input after the ideal latency of the channel model. Each interconnect is simulated for 100,000 code words and errors are measured in terms of code word errors. A code word error occurs when signal errors prevent encoded source data carried by the interconnect from being successfully being decoded. In the case of a non-LHECC enabled interconnect, a code word error occurs whenever any bit within any of the nCm symbols carried by the interconnect is received in error. In the case of an LHECC-enabled interconnect, a code word error occurs whenever there are more bit errors present within the received code word than can be corrected by the code.

Table 7.3 shows three interconnect configurations that are designed to have minimum information and decoding overhead. Each of these interconnects is capable of correcting one nCm symbol that contains a single bit error. In other words, the high-level code consists of a (3, 2, 2, *s*)-checksum that can correct one erasure and each symbol set is partitioned with a distance

of 4. The other two interconnects and associated LHECCs are designed to have higher overhead and multi-symbol error correcting capability. Table 7.4 shows an interconnect configuration that is capable of correcting up to two symbols, each having one bit error. In other words, the high-level code consists of a (4, 2, 3, 3)-MDS code that can correct two erasures and the symbol set is partitioned with a distance of 4. Table 7.5 shows an interconnect configuration that is capable of correcting up to two symbols where each symbol has up to two bit errors if the resultant errors yield invalid nCm symbols. It is also capable of correcting one symbol when the symbol has two bit errors and the resultant errors yield a valid nCm symbol. In other words, the high-level code consists of a (4, 2, 3, 9)-MDS code that can correct two erasures or one error and the symbol set is partitioned with a distance of 6.

In Tables 7.3 – 7.5, the “interconnect” column shows the number and type of MBDS channel. The “width” column shows the overall width of the interconnect in total number of wires. The “capacity (no-ecc)” column shows how many bits of source data is carried by the interconnect assuming a direct, independent mapping from binary to nCm symbols and without LHECC support. 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 is carried in the interconnect when LHECC is utilized, as well as the corresponding absolute code rate. The “rel. code rate” column shows the relative code rate of the interconnect, which is computed as the number of bits carried by the interconnect with LHECC support divided by the number of bits carried by the interconnect without LHECC support. This value represents the amount of relative information overhead required for LHECC. The overhead in bits is the difference in capacity between the non-LHECC enabled interconnect and

the LHECC-enabled interconnect. The results obtained by simulating these interconnects determine the best trade-off point between overhead and error control capability.

Table 7.3: Single-symbol correcting interconnects. The LHECC configuration associated with these interconnects is capable of correcting one symbol containing one bit error.

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

Table 7.4: Multiple-symbol correcting interconnect. The LHECC configuration associated with this interconnect is capable of correcting up to two symbols containing one bit error each.

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

Table 7.5: Multiple-symbol correcting interconnect. 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.

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

In order to measure the performance improvement gained from adding LHECC support to MBDS-based system-level interconnects, the code word error ratio behavior of LHECC interconnects is compared to an equivalent baseline interconnect with no LHECC support. In this case, the baseline interconnect consists of a matching number and type of MBDS channels. Error ratios for both cases are measured over increasing transmission rates. A code word error occurs whenever bit errors exist at the output of the interconnect that cannot be corrected by the decoder. No error correction capability is available in the baseline interconnect, so any bit error that occurs within any of the nCm symbols received from any of the MBDS channels constitutes

a code word error. For an interconnect with LHECC support, a code word error occurs whenever the code word contains types or numbers of errors that cannot be corrected by the LHECC configuration (and corresponding LHECC decoder). For both cases, there is no distinction between detectable and non-detectable errors. The minimum detectable non-zero code word error ratio for these simulations is 10^{-5} based on the simulation run length of 100,000 code words.

8. EXPERIMENTAL RESULTS

This chapter describes results obtained through simulation of the interconnect and noise models described in Chapter 7. The plots shown in this chapter show the performance increase, in terms of code word error ratio, that is gained from adding LHECC support to an MBDS-based interconnect. Code word error ratio is computed as the number of code word errors that occur over a simulation run of 100,000 code words. Due to the simulation run length of 100,000 code words, code word error ratios of less than 10^{-5} do not appear on the plots.

Figure 8.1 shows the results for a single symbol error-correcting LHECC interconnect including only the noise sources that are inherent within the channel model itself, such as transistor switching noise, jitter, and frequency-dependant attenuation and crosstalk within the coupled transmission lines. These plots show that LHECC encoding achieves at least a three order of magnitude decrease in code word error ratio at speeds up to 4.2 GHz for the 4c2 interconnect, up to .4.0 GHz for the 6c3 interconnect, and up to 3.8 GHz for the 8c4 interconnect.

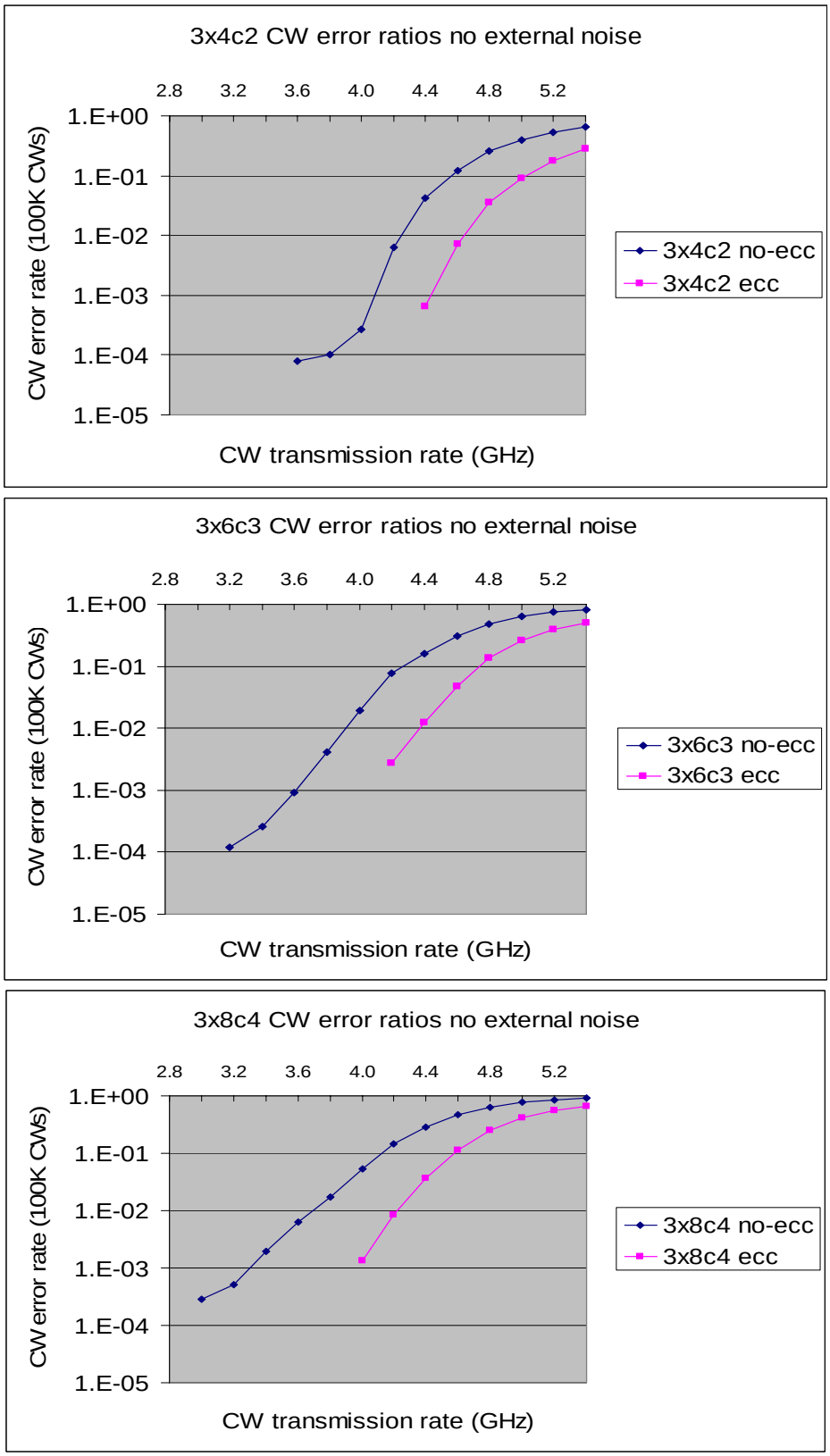


Figure 8.1: Code word error ratios for single-symbol correcting interconnect with no external noise added.

Figure 8.2 indicates the results for a single symbol error-correcting LHECC interconnect with Pentium 4 external supply noise characteristics as described in Chapter 7. In this case, LHECC encoding achieves at least a three order of magnitude drop in error ratio at speeds up to 4.0 GHz for the 4c2 interconnect, up to .3.8 GHz for the 6c3 interconnect, and up to 3.8 GHz for the 8c4 interconnect.

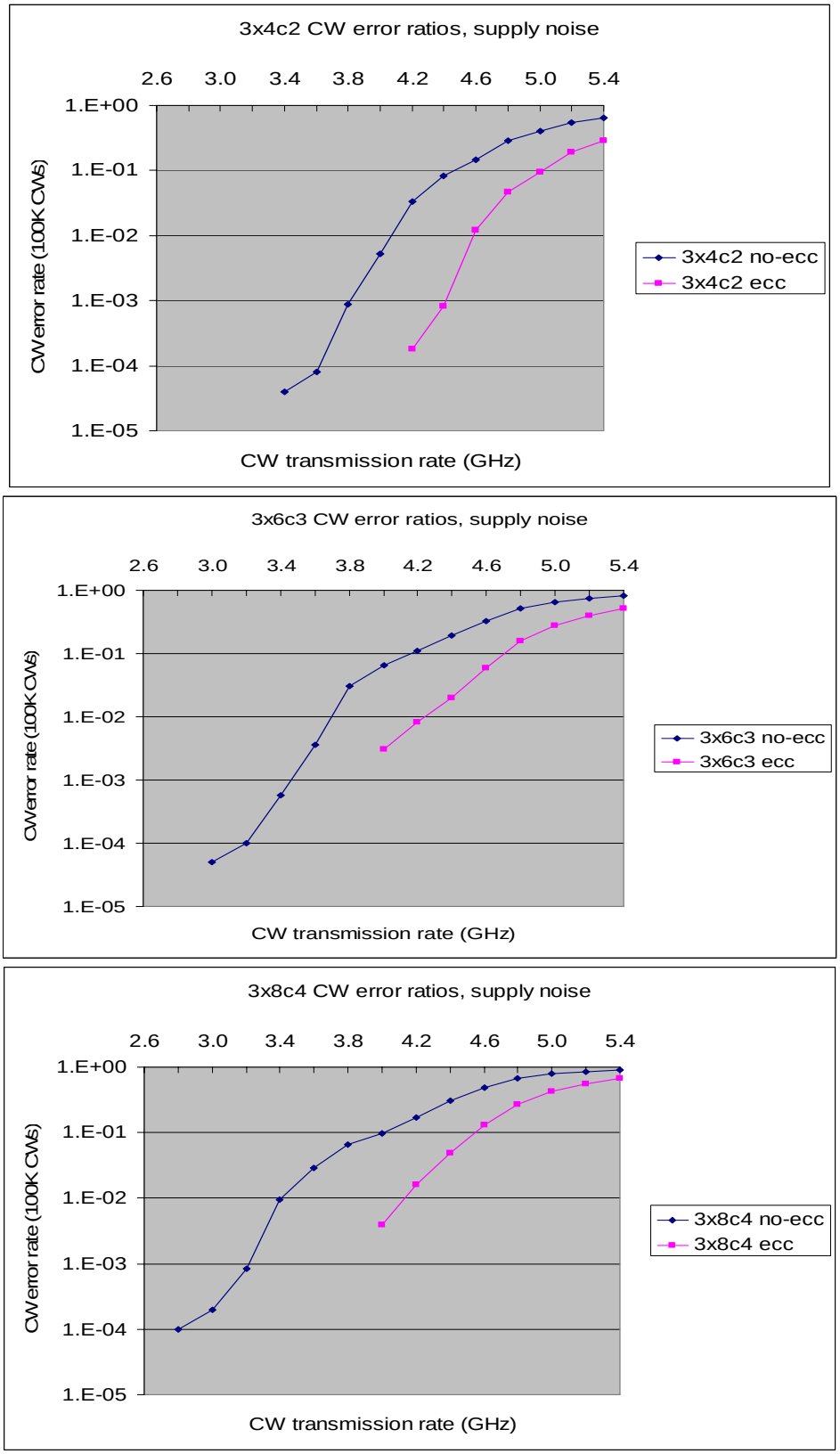


Figure 8.2: Code word error ratios for single-symbol correcting interconnect with supply noise.

Figure 8.3 indicates the results for a single symbol error-correcting LHECC interconnect with additional fringe capacitance and crosstalk. In this case, LHECC encoding achieves at least a three order of magnitude drop in error ratio at speeds up to 3.8 GHz for the 4c2 interconnect, up to .3.6 GHz for the 6c3 interconnect, and up to 3.6 GHz for the 8c4 interconnect.

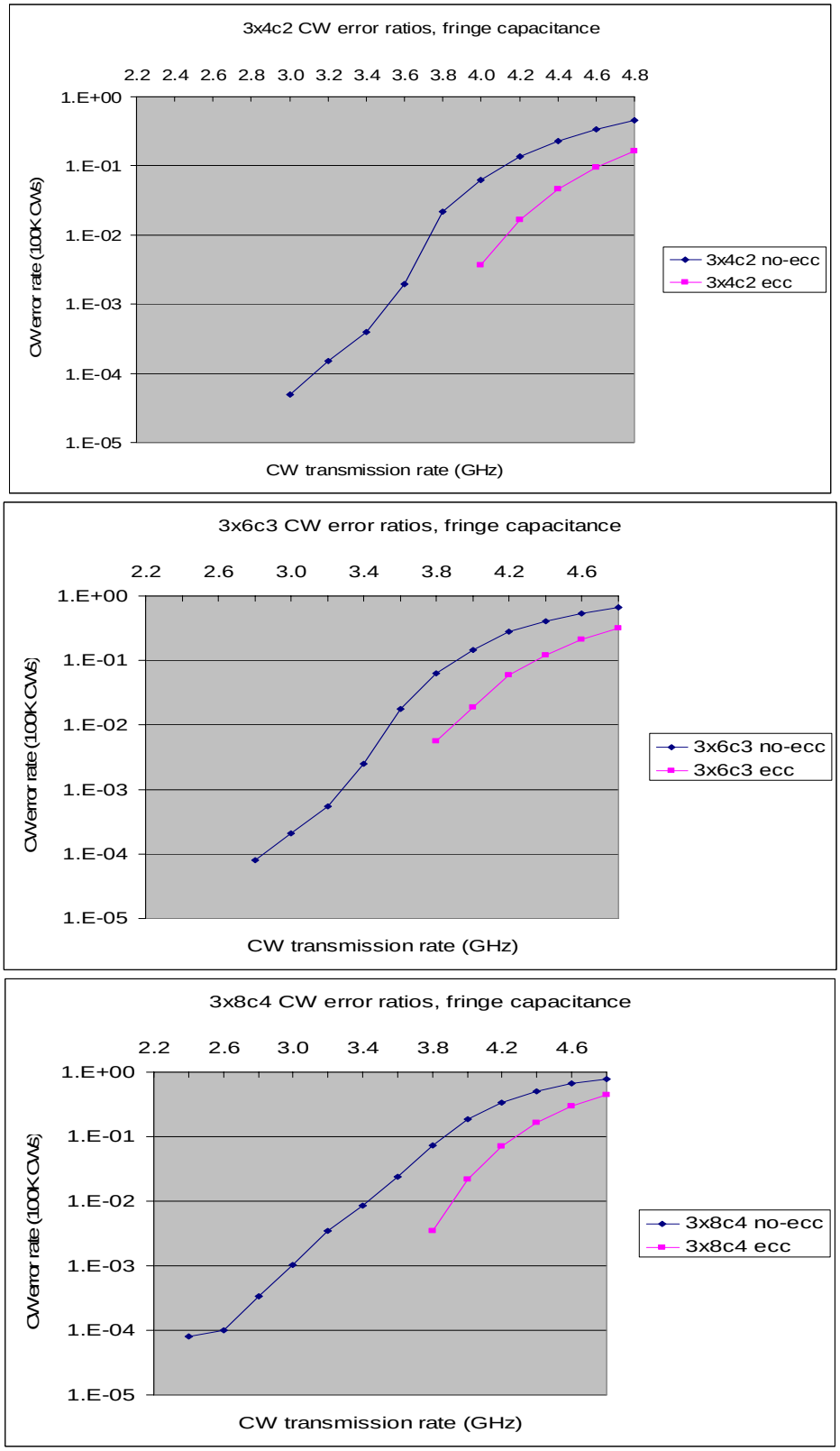


Figure 8.3: Code word error ratios for single-symbol correcting interconnect with fringe capacitance.

Figure 8.4 indicates the results for a double symbol error-correcting LHECC interconnect with channel model noise sources only. In this case, LHECC encoding achieves at least a three order of magnitude drop in error ratio at speeds up to 4.4 GHz for the 4c2 interconnect and up to .4.4 GHz for the 6c3 interconnect.

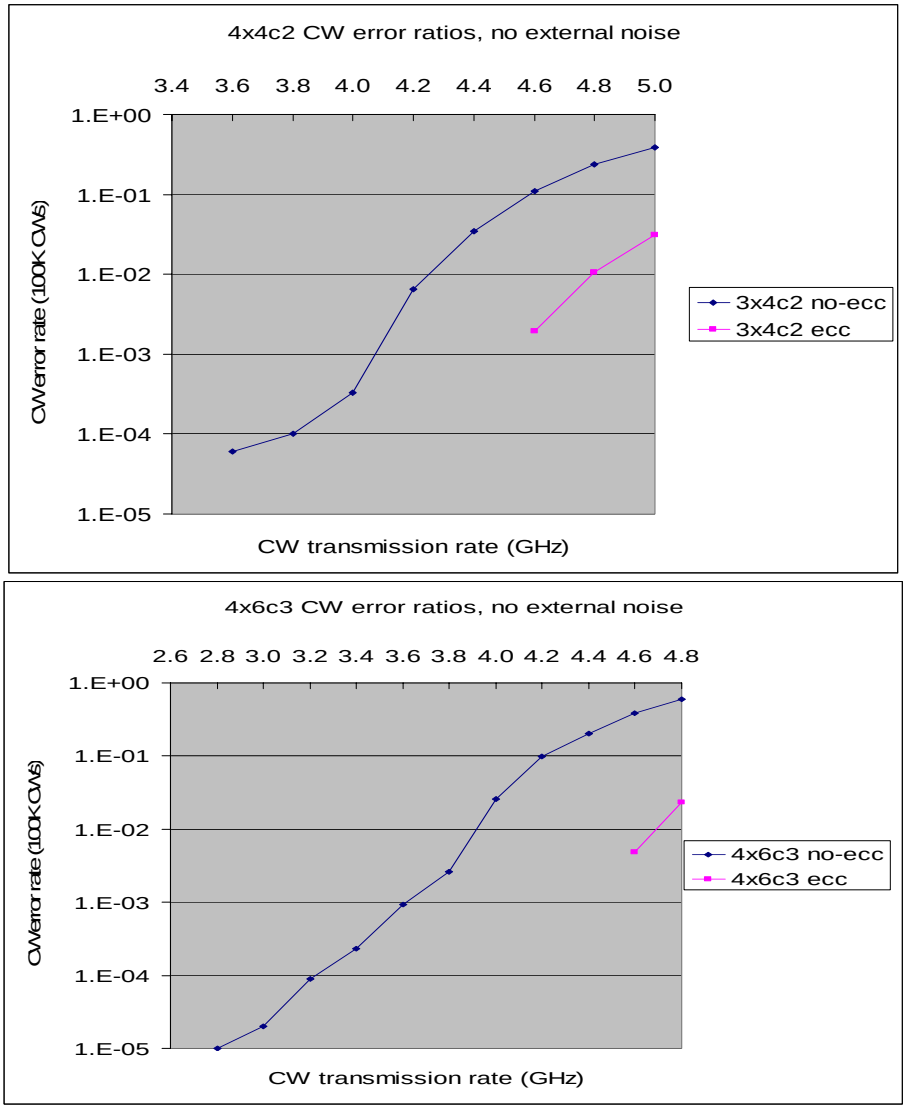


Figure 8.4: Code word error ratios for double-symbol correcting interconnect with no external noise added.

Figure 8.5 indicates the results for a double symbol error-correcting LHECC interconnect with Pentium 4 external supply noise characteristics. In this case, LHECC encoding achieves at least

a three order of magnitude drop in error ratio at speeds up to 4.4 GHz for the 4c2 interconnect and up to .4.2 GHz for the 6c3 interconnect.

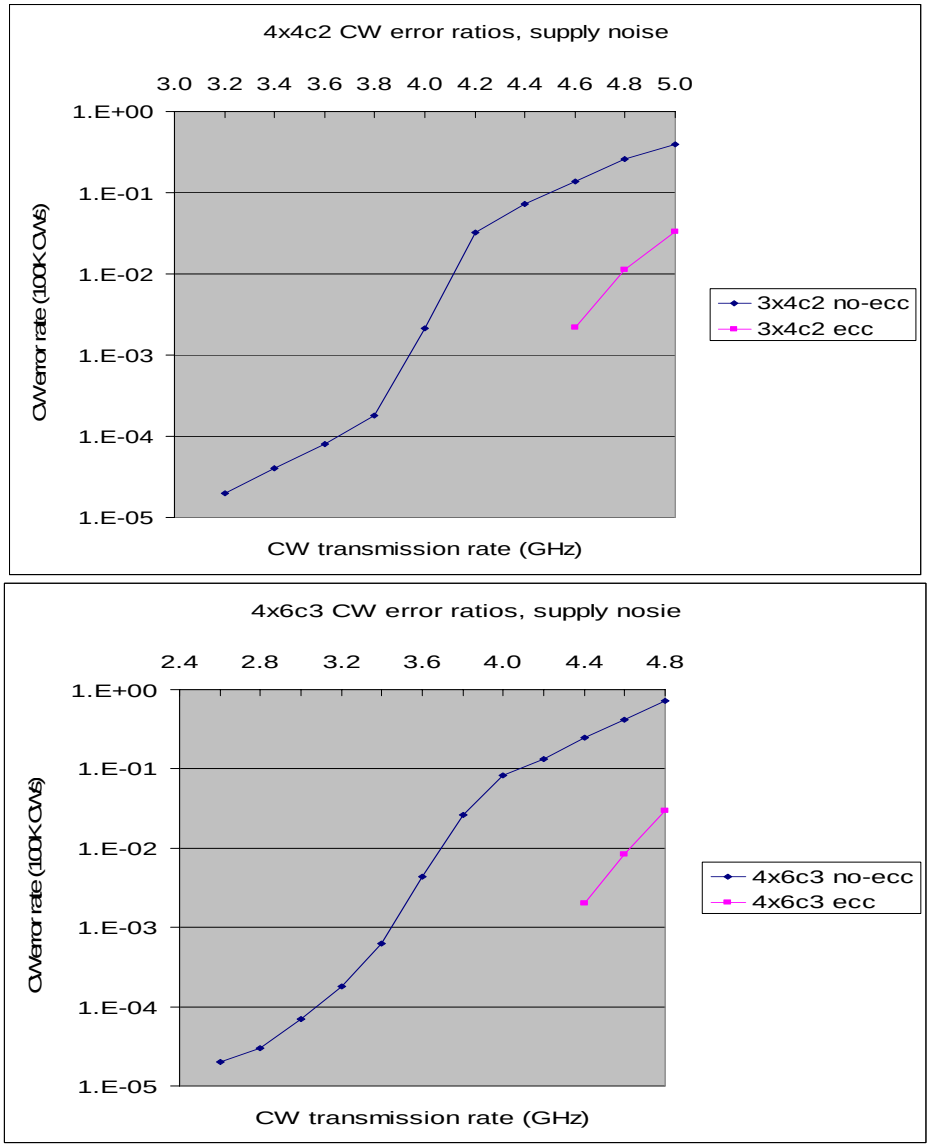


Figure 8.5: Code word error ratios for double-symbol correcting interconnect with supply noise.

Figure 8.6 indicates the results for a double symbol error-correcting LHECC interconnect with additional fringe capacitance and crosstalk. In this case, LHECC encoding achieves at least a three order of magnitude drop in error ratio at speeds up to 3.8 GHz for the 4c2 interconnect and up to 4.0 GHz for the 6c3 interconnect.

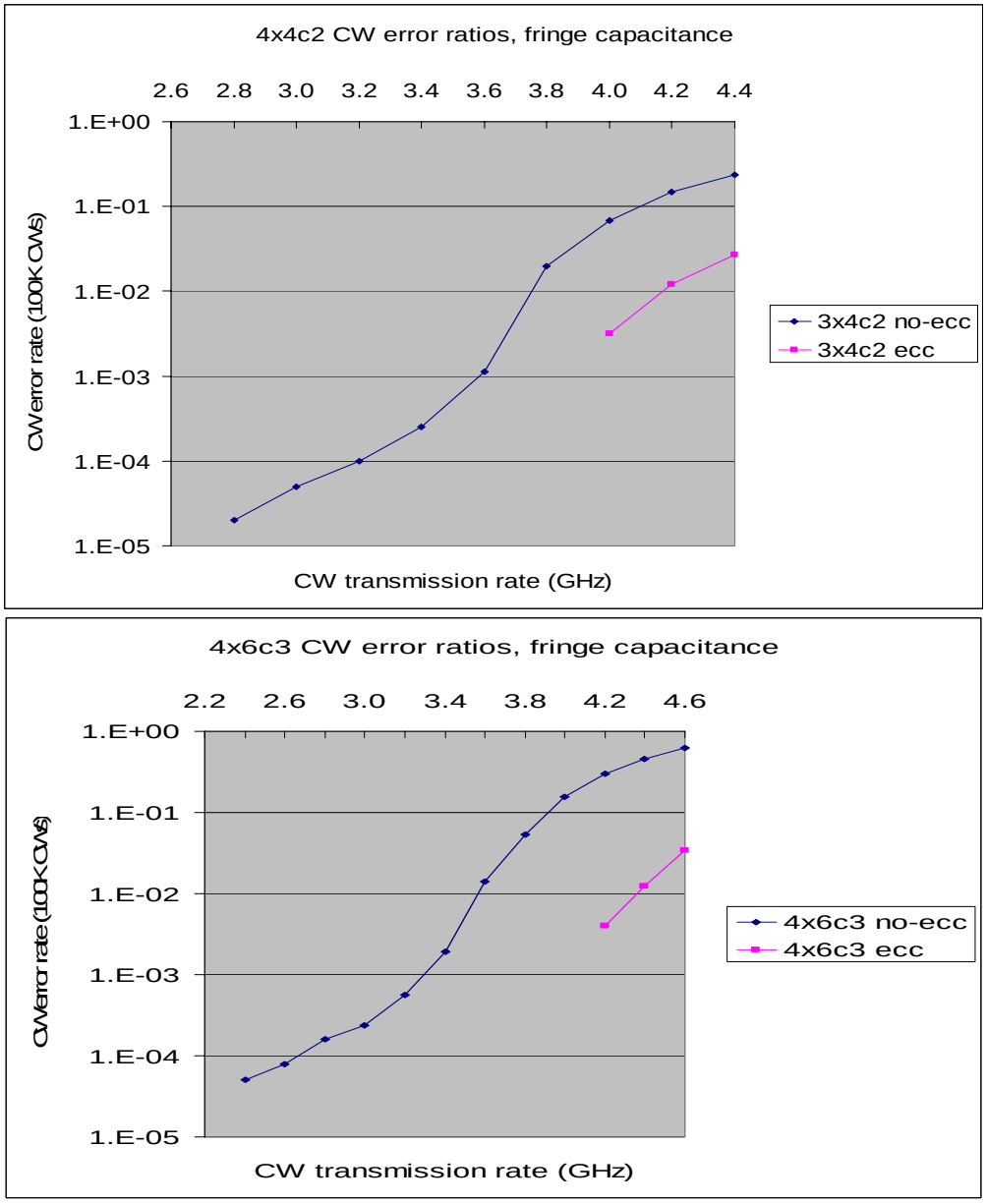


Figure 8.6: Code word error ratios for double-symbol correcting interconnect with fringe capacitance.

9. ANALYSIS

This chapter describes the design parameters that an interconnect designer chooses when designing an LHECC-encoded interconnect formed by parallel MBDS channels. The values for these parameters affect the properties of the resultant code, interconnect, and encoder/decoder complexity.

There are four design parameters that an interconnect designer must set when developing an LHECC configuration. These parameters are outlined below.

- *LHECC block width*

  This parameter defines the number of nCm symbols that make up the LHECC code word. As such, it also defines the width of the high-level code block (in terms of s-data symbols and s-data parity symbols). For this analysis, assume that changing this parameter changes the number of high-level *data symbols* but does not affect the number of high-level *parity symbols*. Increasing this parameter allows more s-data symbols to be encoded into the high-level code and more c-data symbols to be encoded into the low-level code.

- *LHECC symbol width*

  This parameter defines the width of each individual nCm symbol (and thus the nCm symbol set) that makes up the LHECC code word. Changing this parameter forces the code designer to perform a partitioning operation for the new nCm symbol set. For this analysis, assume this new partitioning targets the same Hamming distance as the previous

partitioning. The new partitioning changes the base of both the s-data and the c-data. The change of s-data symbol base forces the selection of a new high-level generator matrix and parity-check matrix to accommodate the new symbol base used in the high-level code.

- *Number of high-level parity symbols*

  This parameter defines the number of parity symbols used in the high-level code. For this analysis, assume that changing this parameter requires that symbols in the high-level code that were originally considered s-data symbols be reassigned as parity symbols. This change also requires that a new generator and parity-check matrix be selected to accommodate the changes in structure of the high-level code.

- *Low-level partitioning distance*

  This parameter defines the Hamming distance of the partitioned subsets. Partitioning an nCm symbol set for a new target Hamming distance requires a change in the base of the s-data and c-data symbols. As such, a new high-level generator and parity-check matrix must be selected to accommodate the new base of the symbols used in the high-level code.

These parameters have effects on three properties of the LHECC configuration. These properties are outlined below.

- *Code rate*

  This property defines the information efficiency of the LHECC code word. It is defined as the data capacity of the code word in bits divided by width of the code word in bits.

- *Encoder/decoder complexity*

This property defines the complexity of the encoder and decoder logic that is required at the end-points of the interconnect. This complexity is measured in total number of standard logic gates.

- *Error control power*

  This property defines the maximum severity of errors that the code is capable of correcting. It is measured in maximum number of symbol errors and maximum number of bit errors per symbol error.

- *Interconnect width*

  This property defines the physical width, in bits or wires, of the LHECC code word and associated interconnect. It is defined as the product of block width and symbol width.

The following subsections describe how each of the design parameters affect the properties of the interconnect.

9.1. Code Rate

Assume the *LHECC block width* is increased, leaving the nCm symbol width, the number of high-level parity symbols, and the low-level partitioning distance fixed. Increasing the block width (by adding additional nCm symbols to the code word) increases the effective code rate. This is because the relative amount of parity information in the high-level code (the ratio of parity symbols to block width) is decreased, and less data is carried by high-level parity symbols as compared to high-level data symbols.

Next, assume that the *nCm symbol width* is increased, leaving the number of high-level parity symbols and the low-level partitioning distance fixed. This change will cause the number of subsets (the base of the s-data symbols) and the subset size (base of the c-data symbols) to increase because wider nCm symbols yield larger symbol sets. Increasing the width of each individual nCm symbol also increases the capacity and effective code rate of the interconnect. This is because each individual nCm symbol carries more data, and wider nCm symbols carry more data per wire than more narrow symbols.

Now, assume the *number of high-level parity symbols* is increased, leaving the nCm symbol width, LHECC block width, and low-level partitioning fixed. The number of parity symbols for the high-level code is increased by reassigning symbols within the high-level block from data symbols to parity symbols. This lowers the interconnect capacity and effective code rate of the interconnect, because less data is carried by the high-level symbols assigned as parity symbols than are carried by the high-level symbols assigned as data symbols. This is because high-level parity symbols carry only c-data symbols, while high-level data symbols carry s-data symbols and c-data symbols. This is shown by the overhead equation, which defines the overhead of any LHECC configuration relative to number of high-level parity symbols (n-k) as floor((n-k)*$\log_2 s$).

Finally, assume that the *low-level partitioning distance* is changed, leaving the LHECC block width, number of high-level parity symbols, and nCm symbol width fixed. An nCm symbol set can be partitioned for any distance, ranging from 4 to the width of the symbol. Partitioning the nCm symbol sets for greater Hamming distance increases the base of the s-data and lowers the

base of the c-data. This causes interconnect capacity and effective code rate to decrease, because the nCm symbols that correspond to the high-level parity symbols carry only c-data symbols, and each of which now represents less information,

9.2. Encoder/Decoder Complexity

Increasing *block width* affects the encoder and decoder complexity in several ways. First, the logic within the encoder and decoder that is responsible for mapping s-data and c-data values to and from nCm symbols scales linearly. This is because one component is required to map each nCm symbol to and from s-data and c-data values, and the requirements of each individual component remains fixed as the number of symbols is increased. Therefore, as additional symbols are added to the code word, additional mapping components are added to the encoder and decoder. Also, the requirement for minimum distance decoding logic remains constant, since only one such component is required for each correctable symbol, and the number of correctable symbols remains fixed along with the number of high-level parity symbols. The complexity of the high-level code components in the encoder and decoder, which include parity generation and high-level block error correction/detection, will increase exponentially. This is because the complexity of these components rises linearly with the size of the high-level code set, and the size of this code set is defined by the size of the message space of the code. This message space equals the total number of combinations of a k-digit message of base q, equaling q^k. In the context of this equation, q (the base of the high-level code symbols) remains constant while k (the number of high-level data symbols representing each s-data message) is increased.

Increasing *nCm symbol width* (and thus the s-data and c-data symbol base) causes the complexity of the encoder and decoder components that are responsible for mapping s-data and c-data values to and from nCm symbols to increase with the size of the symbol set. The set of this symbol set is defined as a function of the width of the symbols (defined by the symbol set size equation in Chapter 3, $n!/((n-m)!*m!)$). In addition, the complexity of the components that perform minimum distance decoding for individual nCm symbols also increases linearly with the product of the width of the nCm symbols and number of subsets. Also, recall that the complexity for the high-level code components is linear to the size of the code set, defined in terms of the high-level message space as q^k. Assuming that the number of symbols forming the block remains fixed, the complexity of the components that generate parity symbols and performs high-level block error correction/detection increases quadratically with number of subsets, which defines the base of the high-level code symbols, q. In this case, the base q (the number of subsets and base of the s-data message symbols) is increased while k (the number of s-data symbols) remains constant.

Increasing the *number of parity symbols* for the high-level code has no affect on the logic complexity of s-data/c-data to nCm to conversion stages. However, additional minimum distance decoders must be added to give the decoder the capability of correcting additional symbols. The complexity of each individual minimum distance decoder remains constant. Therefore the complexity of the low-level code stages in the decoder scales linearly as additional minimum distance decoders are added to the decoder. The high-level code logic will decrease, including parity generation in the encoder and high-level block error correction/detection in the decoder. This is because the complexity of these components is linear to the code set size, and the code set size will decrease with the high-level (s-data) message space (due to the decrease in

total number of high-level data symbols). However, the number of outputs for each of these components will increase. For example, the encoder must now include a component that generates additional high-level parity symbols and the decoder must now include a component that generates additional corrected high-level symbols. However, generation of each parity symbol and corrected symbol may be performed in parallel, since each symbol is computed independently from the others based on the value of the s-data symbols and parity symbols. This means that the complexity increase for generating additional parity symbols and corrected symbols rises linearly.

Increasing the *Hamming distance of each nCm symbol subset* increases the number of subsets and symbol base of the s-data, adding complexity to the high-level code components in the encoder and decoder. Again, because the complexity of the high-level components increases with q^k, increasing the symbol base q while keeping the number of symbols k fixed will cause these components to scale quadratically. The complexity of the minimum distance decoders will increase because the complexity of these components is a linear function of the number of subsets multiplied by the width of the nCm symbol. The complexity of the nCm symbol to c-data/s-data values will remain constant, because the number of nCm symbols will remain constant, while the number of s-data values increases and the number of c-data values decreases.

9.3. Error Correcting Power

Increasing the *LHECC block width* decreases the error correcting power of the code. This is because the code word spans additional nCm symbols, which increases the probability of symbol errors without increasing the number of correctable symbols. Increasing the *nCm symbol width*

also decreases the error correcting power of the code. This is because the code word contains wider nCm symbols, adding more physical bits and more opportunities for any of these bits to be received in error. This increases the probability of additional errors without increasing the number of correctable symbols. Increasing the *number of parity symbols* for the high-level code increases the error correcting capability of the code. This is because additional nCm symbol erasures and errors can be corrected across the entire code block. Increasing the *low-level code partitioning distance* increases the error correcting power of the code, allowing more bit errors to be correctable within individual nCm symbols.

9.4. Rules of Thumb

When designing an LHECC/MBDS interconnect, the designer may choose to maximize error correction power or to maximize code rate and interconnect capacity. For a designer who seeks high error correcting power, the LHECC configuration should include a high low-level partitioning distance and a strong high-level code with multiple parity symbols. This allows for correction of multiple symbol errors that are spread across the code word and across individual nCm symbols. On the other hand, for a designer who seeks high code rate and interconnect capacity, the LHECC configuration should utilize wide nCm symbols, a low-level partitioning distance of 4, and checksum high-level code. This maximizes the amount of data that is encoded into the code word and makes most efficient use of wires. A designer should choose the block size, which dictates overall interconnect width, based on how much logic complexity can be afforded within the system interfaces. Increasing block size while proportionally increasing the number of parity symbols represents a way to increase code rate while keeping error control power constant. However, the block size has the highest impact on encoder/decoder complexity.

10. SUMMARY AND CONCLUSION

This dissertation 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 dissertation illustrates how LHECC is effective at achieving low overhead error control for system-level interconnects built with Multi-Bit Differential Signaling (MDBS) 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, modeling and simulation techniques verify that LHECC improves the performance of interconnects formed with MBDS channels.

LHECC operates over interconnects built with multiple parallel MBDS channels, and each channel requires that transmitted data conform to a fixed symbol set. Each symbol set has unmapped symbol space and an inherent ability for the receiver to detect errors that are local within the symbol. These properties are exploited in the following ways:

- Symbols may be used to carry data and parity information, eliminating the requirement for dedicated parity symbols. This is possible because the symbol set inherently includes error control properties, where at least two bit positions are guaranteed to change as the result of any single symbol change.

- Most types of symbol errors yield invalid symbols and can be detected by the receiver. As a result, erasure decoding techniques may be used and therefore fewer total parity symbols are required in order to achieve symbol correction.
- Symbols that would normally go unused in a direct-to-binary mapping are used to facilitate the symbol set partitioning operation that is used to construct a hierarchical encoding and decoding procedure.
- The hierarchical nature of the encoding and decoding procedure facilitates the design of compact, high-throughput pipelined encoders and decoders. This allows support for these codes to be tightly integrated with system I/O interfaces. The encoders and decoders are thus area efficient and capable of operating at current and projected system interconnect speeds.

Using this technique, code rate that is lost as a result of error control encoding is effectively recovered by using properties of the underlying channel. This tight coupling of error control codes to channel properties can be applied to other types of low code rate channels.

10.1. Contributions and Future Directions

The major contribution of this dissertation is a new technique for tightly coupling an error control code to the properties of an underlying channel technology that has a low code rate (< 1). To the author's knowledge, this form of tight coupling between an error control code and a physical-layer channel code and has never been previously explored. Properties that are specific to low code rate channels are exploited to recover code rate that would otherwise be lost due to error control overhead. In this case, the underlying channel technology is Multi-Bit Differential Signaling (MBDS). However, given the physical limitations that future high-speed channels must overcome, it is likely that future channel technologies will also exhibit low code rate.

The significance of this work is that it can be extended to future channel technologies that have low code rate in order to offset the overhead requirement of error control encoding. For example, while MBDS channels use a physical-layer code to guarantee that the voltage of the common point in the termination network remains fixed, some high-speed serial channels utilize other physical-layer encoding techniques to support clock extraction. For example, the 8 bit / 10 bit encoding used for RapidIO [4] is a symbol set that guarantees minimum transition density and maximum consecutive run lengths of 1-bits or 0-bits (in fact, the 8 bit / 10 bit symbol set consists of the union of the 10c4, 10c5, and 10c6 symbol sets). This type of encoding is similar to the physical-layer code used by MBDS as it has a code rate less than 1 and thus contains inherent error detection capability. As a future investigation, it may be possible to tailor the width and symbol set of such a physical-layer code such that it achieves high effectiveness for clock extraction while simultaneously optimizing it for maximum error control efficiency.

Another future direction is to examine the possibility of applying LHECC techniques to MBDS channels that support *multi-level signaling*. These types of channels are also low code rate channels as compared to other multi-level channel technologies. These channels could theoretically have driver designs and termination networks that match a traditional MBDS channel, but employ a new channel code where any binary combination is possible except all-1 or all-0 symbols. In this case, the voltage of the termination network common point is no longer fixed, but fluctuates depending on the symbol being transmitted. Receivers interpret the differential voltage between each wire and the common point as a signal with multiple discrete voltage levels. The resultant number of voltage levels per wire is not a power of 2. This type of

channel has a higher code rate than a traditional MBDS channel of the same width. For example, a 4-wire channel, when using a 4c1/4c2/4c3 input symbol set, has a total symbol set size of 14 symbols. When using this symbol set, each differential receiver in the channel may sense one of six possible voltages. In this case, there are unused symbols when mapping an 8-bit binary message to a driver that can drive 14 possible symbols. However, this channel has the added property that not 6^4 receiver output combinations are possible given the symbol set of 14 symbols. This property provides additional error detecting power to exploit at the receiver for lightweight error control.

APPENDIX: VHDL SOURCE CODE

This appendix provides VHDL source code (designs) of major components for sample LHECC encoders and decoders. The VHDL code for minor and well-understood components, such as registers, multiplexers, and simple datapath logic, is not shown.

This appendix first provides VHDL that is used for the example encoder and decoder architecture described in Chapter 6 and shown in Figures 6.12 and 6.13. This example is based on a (6c3, 3, 2, 2, 4, 4, 4) LHECC configuration and is applied over an interconnect formed with three parallel 6c3 channels. This LHECC configuration relies on a checksum high-level code and a low-level Hamming distance of 4. It is capable of correcting one nCm symbol with a single bit error. In this example architecture, A and B refer to the two most significant symbols in the code word and S refers to the least significant symbol in the code word. S refers to the "sum," or checksum symbol.

This appendix also provides the VHDL code for three critical components of a (4c2, 4, 2, 3, 3, 2, 4) LHECC, which is operated over an interconnect formed with four parallel 4c2 channels. These components are responsible for MDS encoding, as well as for correction and detection of high-level code errors for its (4, 2, 3, 3)-MDS code. The design of these components is shown to demonstrate the design of an LHECC encoder/decoder that utilizes an MDS-based high-level code. This LHECC configuration is capable of correcting one bit error contained within up to two nCm symbols within the LHECC code word. These components include the "gen. parity,"

"block error detector," and "block corrector" components. In this example architecture, A, B, C, and D refer to the most significant (A), through the least significant (D) symbols forming the code word. C and D encompass the parity symbols relative to the high-level code block.

- Encoder Component for 3 x 6c3 LHECC: **CHECKSUM**

This component accepts two 2-bit s-data values and computes the resultant checksum value.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.all;

ENTITY gen_check_6c3_synth IS
   PORT(
      a0 : IN     std_logic;
      a1 : IN     std_logic;
      b0 : IN     std_logic;
      b1 : IN     std_logic;
      s0 : OUT    std_logic;
      s1 : OUT    std_logic
   );
END gen_check_6c3_synth ;

ARCHITECTURE gen_check_6c3_synth OF gen_check_6c3_synth IS
   signal a : std_logic_vector (1 downto 0);
   signal b : std_logic_vector (1 downto 0);
   signal s : std_logic_vector (1 downto 0);
BEGIN
   a <= a1 & a0;
   b <= b1 & b0;

   s <= a+b;

   s0 <= s(0);
   s1 <= s(1);
END gen_check_6c3_synth;
```

- Encoder Component for 3 x 6c3 LHECC: **GEN. CW**

This component outputs an nCm symbol given an s-data value and c-data value.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
```

```
ENTITY gen_cw_6c3_synth IS
   PORT(
      c0  : IN     std_logic;
      c1  : IN     std_logic;
      s0  : IN     std_logic;
      s1  : IN     std_logic;
      cw0 : OUT    std_logic;
      cw1 : OUT    std_logic;
      cw2 : OUT    std_logic;
      cw3 : OUT    std_logic;
      cw4 : OUT    std_logic;
      cw5 : OUT    std_logic
   );

END gen_cw_6c3_synth ;

ARCHITECTURE gen_cw_6c3_synth OF gen_cw_6c3_synth IS
BEGIN
   gen_cw : process (c0,c1,s0,s1)
   variable c : std_logic_vector (1 downto 0);
   variable s : std_logic_vector (1 downto 0);
   variable cw_out : std_logic_vector(5 downto 0);
   begin
      c := c1 & c0;
      s := s1 & s0;

      if (s="00") and (c="00") then
         cw_out := "000111";
      elsif (s="00") and (c="01") then
         cw_out := "011100";
      elsif (s="00") and (c="10") then
         cw_out := "101010";
      elsif (s="00") and (c="11") then
         cw_out := "110001";
      elsif (s="01") and (c="00") then
         cw_out := "001011";
      elsif (s="01") and (c="01") then
         cw_out := "010110";
      elsif (s="01") and (c="10") then
         cw_out := "100101";
      elsif (s="01") and (c="11") then
         cw_out := "111000";
      elsif (s="10") and (c="00") then
         cw_out := "001101";
      elsif (s="10") and (c="01") then
         cw_out := "011010";
      elsif (s="10") and (c="10") then
         cw_out := "100011";
      elsif (s="10") and (c="11") then
         cw_out := "110100";
      elsif (s="11") and (c="00") then
         cw_out := "001110";
      elsif (s="11") and (c="01") then
         cw_out := "010101";
      elsif (s="11") and (c="10") then
         cw_out := "101001";
```

```
      else
         cw_out := "110010";
      end if;
      cw5 <= cw_out(5);
      cw4 <= cw_out(4);
      cw3 <= cw_out(3);
      cw2 <= cw_out(2);
      cw1 <= cw_out(1);
      cw0 <= cw_out(0);
   end process gen_cw;
END gen_cw_6c3_synth;
```

- Decoder Component for 3 x 6c3 LHECC: **RESOLVER**

This component outputs an s-data component, c-data component, and error flag given an nCm symbol (or nCm symbol that was received in error).

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.all;

ENTITY resolver_6c3_synth IS
   PORT(
      in0   : IN     std_logic;
      in1   : IN     std_logic;
      in2   : IN     std_logic;
      in3   : IN     std_logic;
      in4   : IN     std_logic;
      in5   : IN     std_logic;
      c0    : OUT    std_logic;
      c1    : OUT    std_logic;
      error : OUT    std_logic;
      s0    : OUT    std_logic;
      s1    : OUT    std_logic
   );

END resolver_6c3_synth ;

ARCHITECTURE resolver_6c3_synth OF resolver_6c3_synth IS
   signal i : std_logic_vector(5 downto 0);
   signal s : std_logic_vector(1 downto 0);
   signal c : std_logic_vector(1 downto 0);
BEGIN
   i <= in5 & in4 & in3 & in2 & in1 & in0;

   s <= "00" when (i="000111") or (i="011100") or (i="101010") or
(i="110001")
   else "01" when (i="001011") or (i="010110") or (i="100101") or
(i="111000")
   else "10" when (i="001101") or (i="011010") or (i="100011") or
(i="110100")
```

```
   else "11" when (i="001110") or (i="010101") or (i="101001") or
(i="110010")
   else "--";

   c <= "00" when (i="000111") or (i="001011") or (i="001101") or
(i="001110")
   else "01" when (i="011100") or (i="010110") or (i="011010") or
(i="010101")
   else "10" when (i="101010") or (i="100101") or (i="100011") or
(i="101001")
   else "11" when (i="110001") or (i="111000") or (i="110100") or
(i="110010")
   else "--";

   error <= '0' when (i="000111") or (i="011100") or (i="101010") or
(i="110001")
                or (i="001011") or (i="010110") or (i="100101") or
(i="111000")
                or (i="001101") or (i="011010") or (i="100011") or
(i="110100")
                or (i="001110") or (i="010101") or (i="101001") or
(i="110010")
      else '1';

   s0 <= s(0); s1 <= s(1);
   c0 <= c(0); c1 <= c(1);

END resolver_6c3_synth;
```

- Decoder Component for 3 x 6c3 LHECC: **BLK_CORRECTOR**

This component performs correction for the high-level code block.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.all;

ENTITY halfecc_6c3_synth IS
   PORT(
      ea      : IN     std_logic;
      eb      : IN     std_logic;
      es      : IN     std_logic;
      sa0     : IN     std_logic;
      sa1     : IN     std_logic;
      sb0     : IN     std_logic;
      sb1     : IN     std_logic;
      ss0     : IN     std_logic;
      ss1     : IN     std_logic;
      corr_s0 : OUT    std_logic;
      corr_s1 : OUT    std_logic
   );
```

```
END halfecc_6c3_synth ;

ARCHITECTURE halfecc_6c3 OF halfecc_6c3_synth IS
BEGIN
   truth_process : process (sa1,sa0,sb1,sb0,ss1,ss0,ea,eb,es)

        variable sa : std_logic_vector (1 downto 0);
        variable sb : std_logic_vector (1 downto 0);
        variable ss : std_logic_vector (1 downto 0);

        begin

           if ((ea='1') and (eb='1')) or ((ea='1') and (es='1')) or ((eb='1')
and (es='1')) then

              corr_s1 <= '-';
              corr_s0 <= '-';

           else

              if (ea='1') then

                 sb := sb1 & sb0;
                 ss := ss1 & ss0;
                 sa := ss - sb;

                 corr_s1 <= sa(1);
                 corr_s0 <= sa(0);

              elsif (eb='1') then

                 sa := sa1 & sa0;
                 ss := ss1 & ss0;
                 sb := ss - sa;

                 corr_s1 <= sb(1);
                 corr_s0 <= sb(0);

              else

                 sa := sa1 & sa0;
                 sb := sb1 & sb0;
                 ss := sa + sb;

                 corr_s1 <= ss(1);
                 corr_s0 <= ss(0);

              end if;

        end if;

   END process truth_process;

END halfecc_6c3;
```

- Decoder Component for 3 x 6c3 LHECC: **BLK_ERROR DETECTOR**

This component detects uncorrectable errors. This occurs when there are too many erasures or errors relative to the high-level code block.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.all;

ENTITY checkerror_6c3_synth IS
   PORT(
      ea       : IN     std_logic;
      eb       : IN     std_logic;
      es       : IN     std_logic;
      sa0      : IN     std_logic;
      sa1      : IN     std_logic;
      sb0      : IN     std_logic;
      sb1      : IN     std_logic;
      ss0      : IN     std_logic;
      ss1      : IN     std_logic;
      chkerror : OUT    std_logic
   );

END checkerror_6c3_synth ;

--
ARCHITECTURE checkerror_6c3_synth OF checkerror_6c3_synth IS
   signal sa : std_logic_vector (1 downto 0);
   signal sb : std_logic_vector (1 downto 0);
   signal ss : std_logic_vector (1 downto 0);
BEGIN
      sa <= sa1&sa0;
      sb <= sb1&sb0;
      ss <= ss1&ss0;
      chkerror<='1' when (((ea='0') and (eb='0') and (es='0')) and ((sa+sb)
/= ss)) or (((ea='1') and (eb='1')) or ((ea='1') and (es='1')) or ((eb='1')
and (es='1'))) else
                '0';
END checkerror_6c3_synth;
```

- Decoder Component for 3 x 6c3 LHECC: **MD_DECODER STAGE 1**

This component is the first stage of the minimum distance decoder.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
```

```
ENTITY md_6c3_s1_synth IS
   PORT(
      c0  : IN     std_logic;
      c1  : IN     std_logic;
      c2  : IN     std_logic;
      c3  : IN     std_logic;
      c4  : IN     std_logic;
      c5  : IN     std_logic;
      s0  : IN     std_logic;
      s1  : IN     std_logic;
      xa0 : OUT    std_logic;
      xa1 : OUT    std_logic;
      xa2 : OUT    std_logic;
      xa3 : OUT    std_logic;
      xa4 : OUT    std_logic;
      xa5 : OUT    std_logic;
      xb0 : OUT    std_logic;
      xb1 : OUT    std_logic;
      xb2 : OUT    std_logic;
      xb3 : OUT    std_logic;
      xb4 : OUT    std_logic;
      xb5 : OUT    std_logic;
      xc0 : OUT    std_logic;
      xc1 : OUT    std_logic;
      xc2 : OUT    std_logic;
      xc3 : OUT    std_logic;
      xc4 : OUT    std_logic;
      xc5 : OUT    std_logic;
      xd0 : OUT    std_logic;
      xd1 : OUT    std_logic;
      xd2 : OUT    std_logic;
      xd3 : OUT    std_logic;
      xd4 : OUT    std_logic;
      xd5 : OUT    std_logic
   );

END md_6c3_s1_synth ;

--
ARCHITECTURE md_6c3_s1_synth OF md_6c3_s1_synth IS
BEGIN
      md_decoder : process (c0,c1,c2,c3,c4,c5,s0,s1)
        variable xa : std_logic_vector (5 downto 0);
        variable xb : std_logic_vector (5 downto 0);
        variable xc : std_logic_vector (5 downto 0);
        variable xd : std_logic_vector (5 downto 0);
        variable cw : std_logic_vector (5 downto 0);

        BEGIN

           cw:=c5 & c4 & c3 & c2 & c1 & c0;

           if (s1='0') and (s0='0') then
              xa := cw xor "000111";
              xb := cw xor "011100";
              xc := cw xor "101010";
              xd := cw xor "110001";
```

```
            elsif (s1='0') and (s0='1') then
               xa := cw xor "001011";
               xb := cw xor "010110";
               xc := cw xor "100101";
               xd := cw xor "111000";
            elsif (s1='1') and (s0='0') then
               xa := cw xor "001101";
               xb := cw xor "011010";
               xc := cw xor "100011";
               xd := cw xor "110100";
            else
               xa := cw xor "001110";
               xb := cw xor "010101";
               xc := cw xor "101001";
               xd := cw xor "110010";
            end if;

            xa0 <= xa(0);xa1 <= xa(1);xa2 <= xa(2);xa3 <= xa(3);xa4 <=
xa(4);xa5 <= xa(5);
            xb0 <= xb(0);xb1 <= xb(1);xb2 <= xb(2);xb3 <= xb(3);xb4 <=
xb(4);xb5 <= xb(5);
            xc0 <= xc(0);xc1 <= xc(1);xc2 <= xc(2);xc3 <= xc(3);xc4 <=
xc(4);xc5 <= xc(5);
            xd0 <= xd(0);xd1 <= xd(1);xd2 <= xd(2);xd3 <= xd(3);xd4 <=
xd(4);xd5 <= xd(5);

        END PROCESS md_decoder;
END md_6c3_s1_synth;
```

- Decoder Component for 3 x 6c3 LHECC: **MD_DECODER STAGE 2**

This component is the second stage of the minimum distance decoder.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY md_6c3_s2_synth IS
   PORT(
      xa0      : IN     std_logic;
      xa1      : IN     std_logic;
      xa2      : IN     std_logic;
      xa3      : IN     std_logic;
      xa4      : IN     std_logic;
      xa5      : IN     std_logic;
      xb0      : IN     std_logic;
      xb1      : IN     std_logic;
      xb2      : IN     std_logic;
      xb3      : IN     std_logic;
      xb4      : IN     std_logic;
      xb5      : IN     std_logic;
      xc0      : IN     std_logic;
      xc1      : IN     std_logic;
```

```
      xc2      : IN     std_logic;
      xc3      : IN     std_logic;
      xc4      : IN     std_logic;
      xc5      : IN     std_logic;
      xd0      : IN     std_logic;
      xd1      : IN     std_logic;
      xd2      : IN     std_logic;
      xd3      : IN     std_logic;
      xd4      : IN     std_logic;
      xd5      : IN     std_logic;
      corr_c0  : OUT    std_logic;
      corr_c1  : OUT    std_logic;
      md_error : OUT    std_logic
   );

END md_6c3_s2_synth ;

--
ARCHITECTURE md_6c3_s2_synth OF md_6c3_s2_synth IS

BEGIN
      md_decoder : process (xa0,xa1,xa2,xa3,xa4,xa5,
                            xb0,xb1,xb2,xb3,xb4,xb5,
                            xc0,xc1,xc2,xc3,xc4,xc5,
                            xd0,xd1,xd2,xd3,xd4,xd5)
      variable xa : std_logic_vector (5 downto 0);
      variable xb : std_logic_vector (5 downto 0);
      variable xc : std_logic_vector (5 downto 0);
      variable xd : std_logic_vector (5 downto 0);
      BEGIN
         xa := xa5 & xa4 & xa3 & xa2 & xa1 & xa0;
         xb := xb5 & xb4 & xb3 & xb2 & xb1 & xb0;
         xc := xc5 & xc4 & xc3 & xc2 & xc1 & xc0;
         xd := xd5 & xd4 & xd3 & xd2 & xd1 & xd0;

         if (xa="100000") or (xa="010000") or (xa="001000") or (xa="000100")
or (xa="000010") or (xa="000001") then
            corr_c0 <= '0';
            corr_c1 <= '0';
            md_error <= '0';
         elsif (xb="100000") or (xb="010000") or (xb="001000") or
(xb="000100") or (xb="000010") or (xb="000001") then
            corr_c0 <= '1';
            corr_c1 <= '0';
            md_error <= '0';
         elsif (xc="100000") or (xc="010000") or (xc="001000") or
(xc="000100") or (xc="000010") or (xc="000001") then
            corr_c0 <= '0';
            corr_c1 <= '1';
            md_error <= '0';
         elsif (xd="100000") or (xd="010000") or (xd="001000") or
(xd="000100") or (xd="000010") or (xd="000001") then
            corr_c0 <= '1';
            corr_c1 <= '1';
            md_error <= '0';
         else
            corr_c0 <= '-';
```

```
            corr_c1 <= '-';
            md_error <= '1';
         end if;
      END PROCESS md_decoder;
END md_6c3_s2_synth;
```

- Encoder Component for 4 x 4c2 LHECC: **GEN_PARITY**

This component generates two parity symbols for the MDS-based 4c2 LHECC configuration.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY gen_parity_4c2 IS
   PORT(
      sa : IN     std_logic_vector (1 DOWNTO 0);
      sb : IN     std_logic_vector (1 DOWNTO 0);
      sc : OUT    std_logic_vector (1 DOWNTO 0);
      sd : OUT    std_logic_vector (1 DOWNTO 0)
   );

END gen_parity_4c2 ;

--
ARCHITECTURE gen_parity_4c2 OF gen_parity_4c2 IS
BEGIN
   sc <= "00" when ((sa="00") and (sb="00")) or ((sa="01") and (sb="10")) or
((sa="10") and (sb="01")) else
         "01" when ((sa="00") and (sb="01")) or ((sa="01") and (sb="00")) or
((sa="10") and (sb="10")) else
         "10" when ((sa="00") and (sb="10")) or ((sa="01") and (sb="01")) or
((sa="10") and (sb="00")) else
         "--";

   sd <= "00" when ((sa="00") and (sb="00")) or ((sa="01") and (sb="01")) or
((sa="10") and (sb="10")) else
         "01" when ((sa="00") and (sb="10")) or ((sa="01") and (sb="00")) or
((sa="10") and (sb="01")) else
         "10" when ((sa="00") and (sb="01")) or ((sa="01") and (sb="10")) or
((sa="10") and (sb="00")) else
         "--";
END gen_parity_4c2;
```

- Decoder Component for 4 x 4c2 LHECC: **BLOCK_CORRECTOR**

This component corrects up to two erasures for the high-level code for the MDS-based 4c2 LHECC configuration.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY halfecc_4x4c2 IS
   PORT(
      ea      : IN     std_logic;
      eb      : IN     std_logic;
      ec      : IN     std_logic;
      ed      : IN     std_logic;
      sa      : IN     std_logic_vector (1 DOWNTO 0);
      sb      : IN     std_logic_vector (1 DOWNTO 0);
      sc      : IN     std_logic_vector (1 DOWNTO 0);
      sd      : IN     std_logic_vector (1 DOWNTO 0);
      corr_sa : OUT    std_logic_vector (1 DOWNTO 0);
      corr_sb : OUT    std_logic_vector (1 DOWNTO 0);
      corr_sc : OUT    std_logic_vector (1 DOWNTO 0);
      corr_sd : OUT    std_logic_vector (1 DOWNTO 0)
   );

END halfecc_4x4c2 ;

--
ARCHITECTURE halfecc_4x4c2 OF halfecc_4x4c2 IS
begin
   myproc : process (ea,eb,ec,ed,sa,sb,sc,sd)
   begin
      corr_sa<="--";
      corr_sb<="--";
      corr_sc<="--";
      corr_sd<="--";

      if ea='1' then
         if (eb='0') and (ec='0') then
            if ((sb="00") and (sc="00")) or ((sb="01") and (sc="01")) or
((sb="10") and (sc="10")) then
               corr_sa <= "00";
            elsif ((sb="00") and (sc="01")) or ((sb="01") and (sc="10")) or
((sb="10") and (sc="00")) then
               corr_sa <= "01";
            elsif ((sb="00") and (sc="10")) or ((sb="01") and (sc="00")) or
((sb="10") and (sc="01")) then
               corr_sa <= "10";
            end if;
         elsif (eb='0') and (ed='0') then
            if ((sb="00") and (sd="00")) or ((sb="01") and (sd="10")) or
((sb="10") and (sd="01")) then
               corr_sa <= "00";
            elsif ((sb="00") and (sd="01")) or ((sb="01") and (sd="00")) or
((sb="10") and (sd="10")) then
               corr_sa <= "01";
            elsif ((sb="00") and (sd="10")) or ((sb="01") and (sd="01")) or
((sb="10") and (sd="00")) then
               corr_sa <= "10";
```

```
            end if;
         elsif (ec='0') and (ed='0') then
            if ((sc="00") and (sd="00")) or ((sc="01") and (sd="10")) or
((sc="10") and (sd="01")) then
               corr_sa <= "00";
            elsif ((sc="01") and (sd="01")) or ((sc="10") and (sd="00")) or
((sc="00") and (sd="10")) then
               corr_sa <= "01";
            elsif ((sc="10") and (sd="10")) or ((sc="00") and (sd="01")) or
((sc="01") and (sd="00")) then
               corr_sa <= "10";
            end if;
         end if;

      elsif eb='1' then
         if (ea='0') and (ec='0') then
            if ((sa="00") and (sc="00")) or ((sa="01") and (sc="01")) or
((sa="10") and (sc="10")) then
               corr_sb <= "00";
            elsif ((sa="00") and (sc="01")) or ((sa="01") and (sc="10")) or
((sa="10") and (sc="00")) then
               corr_sb <= "01";
            elsif ((sa="00") and (sc="10")) or ((sa="01") and (sc="00")) or
((sa="10") and (sc="01")) then
               corr_sb <= "10";
            end if;
         elsif (ea='0') and (ed='0') then
            if ((sa="00") and (sd="00")) or ((sa="01") and (sd="01")) or
((sa="10") and (sd="10")) then
               corr_sb <= "00";
            elsif ((sa="00") and (sd="10")) or ((sa="01") and (sd="00")) or
((sa="10") and (sd="01")) then
               corr_sb <= "01";
            elsif ((sa="00") and (sd="01")) or ((sa="01") and (sd="10")) or
((sa="10") and (sd="00")) then
               corr_sb <= "10";
            end if;
         elsif (ec='0') and (ed='0') then
           if ((sc="00") and (sd="00")) or ((sc="01") and (sd="01")) or
((sc="10") and (sd="10")) then
              corr_sb <= "00";
           elsif ((sc="01") and (sd="10")) or ((sc="10") and (sd="00")) or
((sc="00") and (sd="01")) then
              corr_sb <= "01";
           elsif ((sc="10") and (sd="01")) or ((sc="00") and (sd="10")) or
((sc="01") and (sd="00")) then
              corr_sb <= "10";
           end if;
         end if;

      elsif ec='1' then
         if (ea='0') and (eb='0') then
            if ((sa="00") and (sb="00")) or ((sa="01") and (sb="10")) or
((sa="10") and (sb="01")) then
               corr_sc <= "00";
            elsif ((sa="00") and (sb="01")) or ((sa="01") and (sb="00")) or
((sa="10") and (sb="10")) then
```

```
            corr_sc <= "01";
         elsif ((sa="00") and (sb="10")) or ((sa="01") and (sb="01")) or
((sa="10") and (sb="00")) then
            corr_sc <= "10";
         end if;
      elsif (ea='0') and (ed='0') then
         if ((sa="00") and (sd="00")) or ((sa="01") and (sd="10")) or
((sa="10") and (sd="01")) then
            corr_sc <= "00";
         elsif ((sa="00") and (sd="10")) or ((sa="01") and (sd="01")) or
((sa="10") and (sd="00")) then
            corr_sc <= "01";
         elsif ((sa="00") and (sd="01")) or ((sa="01") and (sd="00")) or
((sa="10") and (sd="10")) then
            corr_sc <= "10";
         end if;
      elsif (eb='0') and (ed='0') then
         if ((sb="00") and (sd="00")) or ((sb="10") and (sd="10")) or
((sb="01") and (sd="01")) then
            corr_sc <= "00";
         elsif ((sb="01") and (sd="10")) or ((sb="00") and (sd="01")) or
((sb="10") and (sd="00")) then
            corr_sc <= "01";
         elsif ((sb="10") and (sd="01")) or ((sb="01") and (sd="00")) or
((sb="00") and (sd="10")) then
            corr_sc <= "10";
         end if;
      end if;

   elsif ed='1' then
      if (ea='0') and (eb='0') then
         if ((sa="00") and (sb="00")) or ((sa="01") and (sb="01")) or
((sa="10") and (sb="10")) then
            corr_sd <= "00";
         elsif ((sa="00") and (sb="10")) or ((sa="01") and (sb="00")) or
((sa="10") and (sb="01")) then
            corr_sd <= "01";
         elsif ((sa="00") and (sb="01")) or ((sa="01") and (sb="10")) or
((sa="10") and (sb="00")) then
            corr_sd <= "10";
         end if;
      elsif (ea='0') and (ec='0') then
         if ((sa="00") and (sc="00")) or ((sa="01") and (sc="10")) or
((sa="10") and (sc="01")) then
            corr_sd <= "00";
         elsif ((sa="00") and (sc="10")) or ((sa="01") and (sc="01")) or
((sa="10") and (sc="00")) then
            corr_sd <= "01";
         elsif ((sa="00") and (sc="01")) or ((sa="01") and (sc="00")) or
((sa="10") and (sc="10")) then
            corr_sd <= "10";
         end if;
      elsif (eb='0') and (ec='0') then
         if ((sb="00") and (sc="00")) or ((sb="01") and (sc="10")) or
((sb="10") and (sc="01")) then
            corr_sd <= "00";
```

```
            elsif ((sb="10") and (sc="10")) or ((sb="00") and (sc="01")) or
((sb="01") and (sc="00")) then
               corr_sd <= "01";
            elsif ((sb="01") and (sc="01")) or ((sb="10") and (sc="00")) or
((sb="00") and (sc="10")) then
               corr_sd <= "10";
            end if;
         end if;
      end if;
   end process myproc;
END halfecc_4x4c2;
```

- Decoder Component for 4 x 4c2 LHECC: **BLOCK_ERROR DETECTOR**

This component detects uncorrectable errors relative to the high-level code for the MDS-based 4c2 LHECC configuration.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY checkerror_4x4c2 IS
   PORT(
      sa       : IN     std_logic_vector (1 DOWNTO 0);
      chkerror : OUT    std_logic;
      ea       : IN     std_logic;
      eb       : IN     std_logic;
      ec       : IN     std_logic;
      ed       : IN     std_logic;
      sb       : IN     std_logic_vector (1 DOWNTO 0);
      sc       : IN     std_logic_vector (1 DOWNTO 0);
      sd       : IN     std_logic_vector (1 DOWNTO 0)
   );

END checkerror_4x4c2 ;

--
ARCHITECTURE checkerror OF checkerror_4x4c2 IS
BEGIN
   chkerror<='1' when
(((ea='0') and (eb='0') and (ec='0') and (ed='0')) and
(((sa="00") and (sb="00") and (sc/="00") and (sd/="00")) or
 ((sa="00") and (sb="01") and (sc/="01") and (sd/="10")) or
 ((sa="00") and (sb="10") and (sc/="10") and (sd/="01")) or
 ((sa="01") and (sb="00") and (sc/="01") and (sd/="01")) or
 ((sa="01") and (sb="01") and (sc/="10") and (sd/="00")) or
 ((sa="01") and (sb="10") and (sc/="00") and (sd/="10")) or
 ((sa="10") and (sb="00") and (sc/="10") and (sd/="10")) or
 ((sa="10") and (sb="01") and (sc/="00") and (sd/="01")))) or
 (((ea='1') and (eb='1') and (ec='1')) or ((ea='1') and (eb='1') and
(ed='1')) or ((ea='1') and (ec='1') and (ed='1')) or ((eb='1') and (ec='1')
and (ed='1'))) else
```

```
      '0';
END checkerror;
```

BIBLIOGRAPHY

[1] John Poulton, “Problems and prospects for electrical signaling [VLSI],” Proceedings of the 20th Anniversary Conference on Advanced Research in VLSI, 21-24 March 1999, Page(s): 326.

[2] Stefan Hirsch, Hans-Jörg Pfleiderer, “CMOS receiver circuits for high-speed data transmission according to LVDS-standard,” Proceedings of SPIE Vol. 5117 (2003).

[3] M. Horowitz, C. K. Yang, S. Sidiropoulos, “High-Speed Electrical Signaling: Overview and Limitations,” IEEE Micro, Jan/Feb 1998.

[4] Sauer, C.; Gries, M.; Gomez, J.I.; Weber, S.; Keutzer, K., “Developing a Flexible Interface for RapidIO, Hypertransport, and PCI-Express,” International Conference on Parallel Computing in Electrical Engineering, 2004. PARLEC 2004. 7-10 Sept. 2004 Page(s):129 - 134

[5] Ramin Farjad-Rod, Chih-Kong Ken Yang, Mark Horowitz, Thomas H. Lee, “A 0.4-um CMOS 10-Gb/s 4-PAM Pre-Emphasis Serial Link Transmitter,” IEEE Journal of Solid-State Circuits, Vol. 34, No. 5, May 1999.

[6] Evelina Young, Mark Horowitz, “A 2.4 Gb/s/pin Simultaneous Bidirectional Parallel Link with Per-Pin Skew Compensation,” IEEE Journal of Solid-State Circuits, Vol. 35, No. 11, November 2000.

[7] D. Cecchi, C. Hans, C. Preuss, “A 2 GB/s high speed link with differential simultaneous bi-directional IO,” IEEE Conference on Custom Integrated Circuits, 6-9 May 2000, Pages: 505 – 508.

[8] Intel, “Intel Pentium 4 Procesors on 90 nm Process Datasheet,” www.intel.com.

[9] J. D. Bakos, D. M. Chiarulli, and S. P. Levitan, "Optoelectronic Multi-Chip-Module Implementation of a 64 channel crossbar switch," International Conference of Optics in Computing (OC2002) pp. 161-163, Taipei, Taiwan, April 8 - 11, 2002.

[10] D. Chiarulli, S. Levitan, J. Bakos, "Optoelectronic Multi-Chip Modules," IEEE Mixed Design of Integrated Circuits and Systems, Poland 2003.

[11] Jason Bakos, Donald Chiarulli, and Steven Levitan, "Optoelectronic Multi-Chip Module Demonstrator System" in Optics in Computing, OSA Technical Digest, (Optical Society of America, Washington DC, 2003) pp 117-119.

[12] Donald Chiarulli, Jason Bakos, Leo Selavo, Steven Levitan, John Hansson, Michael Weisser, "Photonic Packaging for Mixed-Technology Sensor Systems," Integrated Photonics Research and Optics in Computing (IPR-OiC’2004), Engelberg, Switzerland, April 21-23, 2004.

[13] Donald M. Chiarulli, Steven P. Levitan, Jason Bakos, Charles Kuznia, "Active Substrates for Optoelectronic Interconnect," IEEE International Symposium on Circuits and Systems, Vancouver, Canada, May 23, 2004.

[14] Jun Ai and Yao Li, "Polymer fiber-image-guide-based embedded optical circuit board," Applied Optics, Vol. 38, No. 2, January 10, 1999.

[15] Yao Li, Ting Wang, Hideo Kosaka, Shigeru Kawai, and Kenichi Kasahara, "Fiber-image-guide-based bit-parallel optical interconnects," Applied Optics, Vol. 35, No. 35, December 10, 1996.

[16] Tomasz Maj, Andrew G. Kirk, David V. Plant, Joseph F. Ahadian, Clifton G. Fonstad, Kevin L. Lear, Karim Tatah, Matthew S. Robinson, and John A. Trezza, "Interconnection of a two-dimensional array of vertical-cavity surface-emitting lasers to a receiver array by means of a fiber image guide," Applied Optics, Vol. 39, No. 5, February 10, 2000.

[17] Xuezhe Zheng, Philippe J. Marchand, Dawei Huang, and Sadik C. Esener, "Free-space parallel multichip interconnection system," Applied Optics, Vol. 39, No. 20, July 10, 2000.

[18] Michael W. Haney, Marc P. Christensen, Predrag Milojkovic, Jeremy Ekman, Premanand Chandramani, Richard Rozier, Fouad Kiamilev, Yue Liu, and Mary Hibbs-Brenner, "Multichip free-space global optical interconnection demonstration with integrated arrays of vertical-cavity surface-emitting lasers and photodetectors," Applied Optics, Vol. 38, No. 29, October 10, 1999.

[19] S. Mick, J. Wilson, P. Franzon, “4 Gbps high-density AC coupled interconnection,” Proceedings of the IEEE Custom Integrated Circuits Conference, May 12-15 2002, Pages:133 – 140.

[20] S. Sidiropoulos, M. Horowitz, “A 700-Mb/s/pin CMOS signaling interface using current integrating receivers,” IEEE Journal of Solid-State Circuits, Volume: 32, Issue: 5, May 5, 1997, Pages: 681 – 690.

[21] B. Young, “An SOI CMOS LVDS driver and receiver pair,”, Digest of Technical Papers. 2001 Symposium on VLSI Circuits, 14-16 June 2001, Pages:153 – 154.

[22] F. Devisch, J. Stiens, R. Vounckx, M. Kuijk, “A power reduction method for off-chip interconnects,” The 2000 IEEE International Symposium on Circuits and Systems, 2000, Volume: 4 , May 28-31, 2000, Pages:265 - 268 vol.4.

[23] A. Faldum, W. Willems, “Codes of Small Defect,” Designs, Codes, and Cryptography, 10, 341-350, 1997. Kluwer Academic Publishers, Boston.

[24] Mario A. De Boer, “Almost MDS Codes,” Designs, Codes, and Cryptography, 9, 143-155, 1996. Kluwer Academic Publishers, Boston.

[25] L. Vanwassenhove, R. Bockstaele, R. Baets, M. Brunfaut, W. Meeus, J. Van Campenhout, J. Hall, H. Melchior, A. Neyer, J. Van Koetsem, R. King, K.Ebeling, P. Heremans, "Demonstration of 2-D Plastic Optical Fibre based optical interconnect between CMOS IC's," Optical Fiber Communication Conference and Exhibit, 2001. OFC 2001, Volume: 3, 2001, Pages: WDD74-1 - WDD74-3 vol.3.

[26] H. Blennemann, Yao-Chao Yang; R. Nikel "Off-chip 400 Mbps signal transmission: noise reduction using non-resonant lengths and other techniques," Multi-Chip Module Conference, 1996. MCMC-96, Proceedings., 1996 IEEE, Feb. 6-7, 1996, Pages:138 – 142.

[27] H. I. Hanafi, R. H. Dennard, C. –L. Chen, R. J. Weiss, D. S. Zicherman, "Design and characterization of a CMOS off-chip driver/receiver with reduced power-supply disturbance," IEEE Journal of Solid-State Circuits, Volume: 27, Issue: 5, May 1992, Pages: 783 – 791.

[28] H. Zhang, V. George, J. M. Rabaey, "Low-swing on-chip signaling techniques: effectiveness and robustness," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Volume: 8 , Issue: 3, June 2000, Pages: 264 – 272.

[29] K. Farzan, D. A. Johns, "Power efficient chip-to-chip signaling schemes," IEEE International Symposium on Circuits and Systems, 2002. ISCAS 2002., Volume: 2, May 26-29, 2002, Pages:II-560 - II-563 vol.2.

[30] A. Bogliolo, “Encodings for high-performance energy-efficient signaling,” International Symposium on Low Power Electronics and Design, 6-7 Aug 6-7, 2001, Pages:170 – 175.

[31] H. Tamura, K. Gotoh, H. Araki, S. Wakayama, T. S. Cheung, M. Saito, J. Ogawa, Y. Kato, T. Nishi, M. Kawano, M. Taguchi, T. Imamura, “PRD-based global-mean-time signaling for high-speed chip-to-chip communications,” Solid-State Circuits Conference, 1998. Digest of Technical Papers. 45th ISSCC 1998 IEEE International, Feb 5-7, 1998, Pages:164 - 165, 432.

[32] Gadiel Seroussi, Ron M. Roth, “On MDS Extensions of Generalized Reed-Solomon Codes,” IEEE Transactions of Information Theory, Vol IT-32, No. 3, May 1986, Pages: 349-354.

[33] John Teifel, Rajit Manohar, “A High-Speed Clockless Serial Link Transceiver,” Proceedings of the Ninth Symposium of Asynchronous Circuits and Systems ASYNC’03, IEEE 2003.

[34] B. W. Langley, R. F. W. Pease, “Superconducting Interconnects for VLSI Multi-Chip System Integration,” Symposium on VLSI Technology, IEEE 1990.

[35] Rambus Corp., “Gigahertz Rambus Signaling Technologies: RSL, QRSL, and SerDes Technology Overview,” http://www.rambus.com.

[36] Randy Mooney (Intel Corp.), “Scaling Methods for Electrical Interconnects to Meet the Performance Requirements of Microprocessor Platforms,” Workshop Notes, IEEE 14th Annual Workshop on Interconnects Within High-Speed Digital Systems, May 4-7, 2003.

[37] C. E. Shannon, “A Mathematical Theory of Communications,” Bell Syst. Tech. J. 27, pp. 379-423 (Part I), 623-656 (Part II), July 1948.

[38] E. R. Berlekamp, “Algebraic Coding Theory,” McGraw-Hill, New York, 1968.

[39] W. W. Peterson, E. J. Weldon, Jr., “Error-Correcting Codes,” 2nd Edition, MIT Press, Cambridge, Mass., 1972.

[40] F. J. MacWilliams, J. J. A. Sloane, “The Theory of Error-Correcting Codes,” North-Holland, Amsterdam, 1977.

[41] R. J. McEliece, “The Theory of Information and Coding,” Addison-Wesley, Reading, Mass., 1977.

[42] F. J. MacWilliams, “A Theory on the Distribution of Weights in a Systematic Code,” Bell Sys. Tech. J., 42, pp. 79-94, 1963.

[43] R. W. Hamming, “Error Detecting and Error Correcting Codes,” Bell Sys. Tech. J.,29, pp. 147-160, April 1950.

[44] S. K. Leung-Yan-Cheong, M. E. Hellman, “Concerning a Bound on Undetected Error Probability,” IEEE Transactions on Information Theory, IT-22, pp. 235-237, March 1976.

[45] E. Prange, “Cyclic Error-Correcting Codes in Two Symbols,” AFCRC-TN-57, 103, Air Force Cambridge Research Center, Cambridge, Mass., September 1957.

[46] T. Kasami, “A Decoding Method for Multiple-Error-Correcting Cyclic Codes by using Threshold Logics,” Conv. Rec. Inf. Process. Soc. Jap. (in Japanese), Tokyo, November 1961.

[47] M. E. Mitchell et al, “Coding and Decoding Operation Research,” G. E. Advanced Electronics Final Report on Contract AF 19 (604)-6183, Air Force Cambridge Research Labs., Cambridge, Mass., 1961.

[48] M. E. Mitchell, “Error-Trap Decoding of Cyclic Codes,” G. R. Report No. 62MCD3, G. E. Military Communications Dept., Oklahoma City, Okla., December 1962.

[49] L. Rudolph, “Easily Implemented Error-Correction Encoding-Decoding,” G. E. Report No. 62MCD2, G. E. Corporation, Oklahoma City, Okla., December 1962.

[50] T. Kasami, “A Decoding Procedure For Multiple-Error-Correction Codes,” IEEE Transactions on Information Theory, IT-10, pp. 134-139, April 1964.

[51] A. Hocquenghem, “Codes corecteurs d’erreurs,” Chiffres, 2, pp. 147-156, 1959.

[52] R. C. Bose, D. K. Ray-Chaudhuri, “On a Class of Error Correcting Binary Group Codes,” Information Control, 3, pp. 68-69, March 1960.

[53] W. W. Peterson, “Encoding and Error-Correction Procedures for the Bose-Chaudhuri-Codes,” IRE Transactions on Information Theory, IT-6, pp. 459-470, September 1960.

[54] D. Gorenstein, N. Zierler, “A Class of Cyclic Linear Error-Correcting Codes in pm Symbols,” J. Soc. Ind. Appl. Math., 9, pp. 107-214, June 1961.

[55] R. T. Chien, “Cyclic Decoding Procedure for the Bose-Chaudhuri-Hocquenghen Codes,” IEEE Transactions on Information Theory, IT-10, pp 357-363, October 1965.

[56] G. D. Forney, “On Decoding BCH Codes,” IEEE Transactions on Information Theory, IT-11, pp. 549-557, October 1965.

[57] E. R. Berlekamp, “On Decoding Bose-Chaudhuri-Hocquenghen Codes,” IEEE Transactions on Information Theory, IT-11, pp. 577-580, October 1965.

[58] E. R. Berlekamp, “Algebraic Coding Theory,” McGraw-Hill, New York, 1968.

[59] J. L. Massey, “Step-by-Step Decoding of Bose-Chaudhuri-Hocquenghen Codes,” IEEE Transactions on Information Theory, IT-11, pp. 580-585, October 1965.

[60] J. L. Massey, “Shift-Register Synthesis and BCH Decoding,” IEEE Transactions on Information Theory, IT-15, pp.122-127, January 1969.

[61] H. O. Burton, “Inversionless Decoding of Binary BCH Codes,” IEEE Transactions on Information Theory, IT-17, pp. 464-466, July 1971.

[62] I. S. Reed, G. Solomon, “Polynomial Codes over Certain Finite Fields,” J. Soc. Ind. Appl. Math., 8, pp. 300-304, June 1960.

[63] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, “Efficient Erasure Correcting Codes,” IEEE Transactions on Information Theory, Vol. 47, No. 2, February 2001.

[64] R. M. Roth, G. Seroussi, “Location-Correcting Codes,” IEEE International Symposium on Information Theory, Trondheim, Norway, June 1994.

[65] G. D. Forney, Jr., “Concatenated Codes,” MIT Press, Cambridge, Mass., 1966.

[66] T. V. Ramabadran, “A Coding Scheme for m-out-of-n Codes,” IEEE Transactions on Communications, Vol. 38, No. 8, August 1990, pp. 1156-1163.

[67] A. Burr, “Turbo-Codes: the utimate error control codes?,” IEEE Electronics and Communication Engineering Journal, August 2001, pp. 155-165.

[68] T. H. Szymanski, “Optical Link Optimization Using Embedded Forward Error Correcting Codes,” IEEE Journal of Selected Topics in Quantum Electronics, Vol. 9, No. 2, March/April 2003, pp. 647-656.

[69] W. F. Chang, C. W. Wu, “Low-Cost Modular Totally Self-Checking Checker Design for m-out-of-n Code,” IEEE Transactions on Computers, Vol. 48, No. 8, August 1999, pp. 815-826.

[70] C. Bolchini, F. Salice, D. Sciuto, “Conditions for the Design of Circuits with Concurrent Error Detection Properties,” 1997 IEEE International Symposium on Circuits and Systems, June 9-12, 1997, Hong Kong, pp. 2741-2744.

[71] J. C. Lo, “A Hyper Optimal Encoding Scheme for Self-Checking Circuits,” IEEE Transactions on Computers, Vol. 45, No. 9, September 1996, pp. 1022-1030.

[72] A. J. Goor, M. S. Abadir, A. Carlin, “Minimal Test for Coupling Faults in Word-Oriented Memories,” Proceedings of the 2002 Design, Automation, and Test in Europe Conference and Exhibition, (DATE’02).

[73] X. M. Wang, Y. X. Yang, “On the Undetected Error Probability on Nonlinear Binary Constant Weight Codes,” IEEE Transactions on Communications, Vol. 42, No. 7, July 1994, pp. 2390-2394.

[74] F. W. Fu, T. Klove, “The Undetected Error Probability Threshold of m-out-of-n Codes,” IEEE Transactions on Information Theory, Vol. 46, No. 4, July 2000, pp. 1597-1599.

[75] Y. Shin, K. Choi, Y. H. Chang, “Narrow Bus Encoding for Low-Power DSP Systems,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 9, No. 5, October 2001.

[76] M. Stan, W. P. Burleson, “Low-Power Encodings for Global Communication in CMOS VLSI,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 5, No. 4, December 1997.

[77] http://en.wikipedia.org/wiki/LVDS.

[78] M. N. O. Sadiku, “Elements of Electromagnetics Third Edition,” Oxford University Press.

[79] P. Elias, “Coding for Noisy Channels,” IRE Conv. Rec., Part 4, pp. 37-47, 1955.

[80] A. J. Viterbi, “Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm,” IEEE Transactions on Information Theory, IT-13, pp. 260-269, April 1967.

[81] 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, B.K. Gilbert, “System-level Modeling and Simulation of the 10G Optoelectronic Interconnect,” IEEE Journal of Selected Topics in Quantum Electronics, Vol. 21, No. 12, pp.3244-3256, December 2003.

[82] Private correspondence with Martin Saint-Laurent, Intel Corporation.

[83] D. Bertozzi, L. Benini, “Xpipes: a network-on-chip architecture for gigascale systems-on-chip,” Circuits and Systems Magazine, IEEE, Volume 4, Issue 2, 2004 Page(s): 18 – 31.

[84] S. Lin and D. J. Costello, Error Control Coding: Fundamental and Applications: Prentice-Hall, 1983.

[85] Data sheet for Comtech AHA Corporation’s AHA4012B 1.5 MB/s Reed-Solomon Error Correction Device.

[86] http://en.wikipedia.org/wiki/Convolutional_code.

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

[88] Donald M. Chiarulli, Jason D. Bakos, Joel R. Martin, Steven P. Levitan, Area, Power, and Pin Efficient Bus Transceiver Using Multi-Bit-Differential Signaling, IEEE International Symposium on Circuits and Systems (ISCAS2005), Kobe, Japan, May 23-26, 2005.

[89] Donald M. Chiarulli, Jason Bakos, Joel R. Martin, Steven P. Levitan, Area, power, and pin efficient bus structures using multi-bit-differential signaling, SPIE Europe International Symposium: Microtechnologies for the New Millennium, Sevilla, Spain 9-11 May 2005.

[90] Steven P. Levitan, Donald M. Chiarulli, Sam Dickerson, Jason Bakos, Joel Martin, Power Efficient Communication Using Multi-Bit-Differential Signaling, 16th Annual IEEELEOS Workshop on Interconnections within High-Speed Digital Systems, 8–11 May 2005.

[91] Highland Communications Technologies Data Sheet, http://www.highlandcomm.com/rs_codes/HLCT_FLEXRS_HPT_DS.pdf.

[92] Laboratory and Flight Performance of the Mars Pathfinder (15,1/6) Convolutionally Encoded Telemetry Link, NASA, http://tmo.jpl.nasa.gov/tmo/progress_report/42-129/129I.pdf.

[93] Data sheet for Comtech AHA Corporation’s AHA4541 Mbit/s Turbo Coding Chip

[94] Xilinx Corporation, Reed-Solomon Decoder LogicCore v5.1. http://www.xilinx.com/ipcenter/catalog/logicore/docs/rs_decoder.pdf.

[95] Sunghoon Kwon; Hyunchul Shin, “An area-efficient VLSI architecture of a Reed–Solomon decoder/encoder for digital VCRs,” IEEE Transactions on Consumer Electronics, Volume 43, Issue 4, Nov. 1997 Page(s):1019 – 1027.

[96] Wilhelm, W., “A new scalable VLSI architecture for Reed-Solomon decoders,” IEEE Journal of Solid-State Circuits, Volume 34, Issue 3, March 1999 Page(s):388 – 396.

[97] Hsie-Chia Chang; Shung, C.B.; Chen-Yi Lee, “A Reed-Solomon product-code (RS-PC) decoder chip for DVD applications,” IEEE Journal of Solid-State Circuits, Volume 36, Issue 2, Feb. 2001 Page(s):229 – 238.

[98] Hanho Lee, “High-speed VLSI architecture for parallel Reed-Solomon decoder,” Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03, Volume 2, 25-28 May 2003 Page(s):II-320 - II-323 vol.2.

[99] Jah-Ming Hsu; Chin-Liang Wang, “An area-efficient pipelined VLSI architecture for decoding of Reed-Solomon codes based on a time-domain algorithm,” IEEE Transactions on Circuits and Systems for Video Technology, Volume 7, Issue 6, Dec. 1997 Page(s):864 – 871.

[100] Hsie-Chia Chang; Chien-Ching Lin; Chen-Yi Lee, “A low-power Reed-Solomon decoder for STM-16 optical communications,” Proceedings of the 2002 IEEE Asia-Pacific Conference on ASIC, 2002, 6-8 Aug. 2002 Page(s):351 – 354.

[101] Xu Yuanxin; Xia Fang; Yao Qingdong; Qiu Peiliang; Wang Kuang, "A new VLSI design for decoding of Reed-Solomon codes based on ASIP," Proceedings of the 4th International Conference on ASIC, 2001, 23-25 Oct. 2001 Page(s): 448 - 451.