A Level-Encoded Transition Signaling Protocol for High-Throughput Asynchronous Global Communication

Peggy B. McGee, Melinda Y. Agyekum, Moustafa M. Mohamed and Steven M. Nowick

{pmcgee, melinda, mmohamed, nowick}@cs.columbia.edu

Department of Computer Science
Columbia University

April 10, 2008
Trends in Digital Systems Design

- **Increased design complexity**
  - More functionality on a single chip
    - Smaller transistor size
    - Larger die size
  - Multiple clock domains

- **High-performance computing**
  - Multi-Giga Hertz clock rate
  - Multiple independent computation nodes
    - Processor cores, memories, etc.

- **Plug-&-play components**
  - For re-usability

System-on-Chip (SoC)
System-on-Chip (SoC): Challenges

- **Heterogeneity**
  - Multiple clock domains
  - Mixed asynchronous/synchronous components

- **Wires do not scale at the same rate as transistors**
  - Increasing proportion of delay in interconnects
  - Challenges for global routing in physical design

- **Deep submicron effects**
  - Handling dynamic timing variability, crosstalk, EMI, noise, etc.
  - Clock jittering and/or drifting effects

- **Power dissipation**
  - Interconnects a significant source of power

*Need for new approaches for interconnect design*
SoC Communication Fabric: Ideal Requirements

- **Speed**
  - High throughput, low latency

- **Low power**
  - Low switching activity

- **Robustness**
  - Against timing variation
  - Handling dynamic voltage scaling
  - Handling single-event upset effects (soft errors)

- **Flexibility**
  - Easy integration of modular Intellectual Properties (IPs)
Asynchronous Design for SoC Communication

Potential benefits of asynchronous design

- Significant power advantage
  - No clock routing
  - “Compute-on-demand” approach

- Timing robustness using delay-insensitive (DI) encoding
  - Eliminates global timing constraints
  - Accommodates uncertainties in routing delay
  - Accommodates skew between bits

- Supports modular design methodologies
  - e.g. GALS (globally-asynchronous, locally-synchronous)
  - Mixed synchronous/asynchronous components

Asynchronous design well-suited for ideal requirements of SoC communication
Application Model: Target SoC Architecture

Our focus

Asynchronous communication channel

Data encode or decode

Computation node
Asynchronous / Synchronous

Data encode or decode

Computation node
Asynchronous / Synchronous

Data encode or decode

Computation node
Asynchronous / Synchronous

Data encode or decode

Computation node
Asynchronous / Synchronous

Data encode or decode

Computation node
Asynchronous / Synchronous

Data encode or decode
Application Model: Target SoC Architecture

1. Timing-robust, high-throughput asynchronous encoding scheme

Our focus

Asynchronous communication channel
Application Model: Target SoC Architecture

1. Timing-robust, high-throughput asynchronous encoding scheme

2. Protocol conversion interface
   → Allows separation of computation and communication
     • Some codes are better for computation
     • Some codes are better for communication

Our focus

Asynchronous communication channel
**Application Model: Target SoC Architecture**

Current focus is on asynchronous computation nodes → Expandable to synchronous
Key Contributions: Theoretical

- A new class of delay-insensitive code for global communication

“Level-Encoded Transition Signaling (LETS)”

- Delay-insensitive
  → Timing-robust

- Uses two-phase (transition) signaling
  → High throughput: no return-to-zero phase
    → most existing schemes use four-phase: have spacer phase
  → Low switching activity

- Level-encoded data
  → Data values easily extracted from encoding

- Supports 1-of-N encoding
  → Lower switching activity
    → compared to existing level-encoded transition signaling code
  → Main focus: 1-of-4 codes
Key Contributions: Practical

- **Practical 1-of-4 LETS codes**
  - Two example codes shown
    - “Quasi-1-hot/cold”
    - “Quasi-binary”

- **Generalization to 1-of-N LETS codes**
  - First to demonstrate 1-of-N level-encoded codes
  - Systematic procedure to generate LETS codes for all $N = 2^n$

- **Hardware support**
  - Efficient conversion circuit for 1-of-4 LETS proposed
    - To/from 4-phase dual-rail signaling
  - Pipeline design for global communication proposed
    - Improves throughput
Outline

- Introduction
- **Background**
  - *Handshake protocol control signaling*
  - *Handshake protocol: control signaling + data*
  - *Asynchronous data encoding*
- 1-of-4 LETS codes
- 1-of-N LETS codes
- Hardware support
- Analytical evaluation
- Conclusions
Handshake Protocol Control Signaling: 4-Phase

- **Four** wire transition events per transaction
- All wires must **return to zero**
  → Before next transaction
Handshake Protocol Control Signaling: 2-Phase

- Two wire transition events per transaction
- No return-to-zero phase
Handshake Protocol: Control Signaling + Data

Sender

Data wire

Receiver

Control = Ack
Handshake Protocol: Control Signaling + Data

Sender

Data

Receiver
Handshake Protocol: Control Signaling + Data

Entire data wave arrives
Handshake Protocol: Control Signaling + Data

Entire data wave arrives

Sender

Receiver

Receiver sends Ack
**Handshake Protocol: Control Signaling + Data**

Entire data wave arrives

2-phase transition signaling protocol completes

→ Transition signaling = non-return-to-zero (NRZ)
Handshake Protocol: Control Signaling + Data

Spacer tokens \((\text{spacer} = \text{data reset to zero})\)

Round trip for 4-phase (return-to-zero) protocol
Handshake Protocol: Control Signaling + Data

All wires reset to zero

Sender

Receiver

Receiver sends Ack

4-phase (return-to-zero) protocol completes
Asynchronous Data Encoding: DI Codes

Properties of delay-insensitive (DI) codes

- Timing-robust
  - Insensitive to input arrival time

- Completion of data transaction encoded into data itself
  - Unambiguous recognition of code
    - no valid codeword seen when transitioning between codewords
DI Return-to-Zero (RZ) Code #1: Dual-Rail

- Two wires to encode a single bit

\[
\begin{align*}
\hat{a} & \quad \quad \quad \quad \quad \quad \quad \quad \quad a_0 \\
\text{(1 bit of data)} & \quad \quad \quad \quad \quad \quad \quad \quad \quad a_1
\end{align*}
\]

Each dual-rail pair provides

- **Data value**: whether 1 or 0 is being transmitted
- **Data validity**: whether data is a value, illegal or reset

**Main benefit**: allows simple hardware for computation blocks

**Main disadvantage**: low throughput and high power

→ Needs reset phase: all bits always reset to zero

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Symbolic value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_1)</td>
<td>(a_0)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
DI Return-to-Zero (RZ) Code #2: 1-of-N

- **N wires to encode log N bits (one-hot encoding)**

Example: 1-of-4 code

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Symbolic value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a3</td>
<td>a2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Main benefit:** uses lower power than dual-rail
  → 1 out of N rails changes value per data transaction

- **Main disadvantage:** gets expensive beyond 1-of-4
  → Coding density decrease
  → Complicated to concatenate irregularly-sized data streams
**DI Non-Return-to-Zero (NRZ) Code #1: LEDR**

**LEDR = Level-Encoded Dual-Rail**

- **Two wires to encode a single bit**

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Symbolic value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase</td>
<td>Parity rail</td>
</tr>
<tr>
<td>Even</td>
<td>0</td>
</tr>
<tr>
<td>Odd</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Properties of LEDR codes:**
  - *Level encoded:* can retrieve data value directly from wires
  - *Alternating phase protocol:* between odd and even phases
  - *Only 1 rail changes value:* per bit per data transaction

**Main benefits**
- No return-to-zero phase
  - High throughput, low power
- Easy to extract data

**Main disadvantages**
- **Significantly** more complicated function blocks
  - No practical solutions have been proposed
  - Potential solution strategy:
    - LEDR for *global communication*
    - 4-phase RZ (dual-rail or single-rail) for *computation*
  - Need efficient hardware for conversion between protocols:
    - Mitra, McLaughlin and Nowick, “Efficient asynchronous protocol converters for two-phase delay-insensitive global communication”, ASYNC’07
- **Uses more power than synchronous communication**
  - Uses less power than RZ
Outline

- Introduction
- Background
- 1-of-4 LETS codes
- 1-of-N LETS codes
- Hardware support
- Analytical evaluation
- Conclusions
LETS Codes: Motivation & Contributions

“LETS = Level-Encoded Transition Signaling”

- **A new class of delay-insensitive codes**
  - Extension of LEDR = 1-of-2 LETS
    - Uses fewer wire transitions per data transaction
    - Analogous to 1-of-N extension to dual-rail in RZ
  - **Goal:**
    - Generate and evaluate entire family of 1-of-N codes

- **Key benefits**
  - Maintains benefits of LEDR
    - High throughput
    - Delay-insensitive
    - Efficient hardware conversion to 4-phase protocols
  - **Additional benefit**
    - Lower power consumption than LEDR
1-of-4 LETS Code Derivation: Overview

Starting point: 4-bit code space

Code space represented by 4-D hypercube

16 codewords in code space
**Goal:** assign symbols to codewords

→ **Symbols** to assign = \{S_0, S_1, S_2, S_3\}
→ **Codewords** = \{0000, 0001, ..., 1111\}

→ such that all LETS properties are observed
**1-of-4 LETS Code Derivation: Overview**

**Goal:** assign symbols to codewords

- **Symbols** to assign = \{S0, S1, S2, S3\}
- **Codewords** = \{0000, 0001, ...., 1111\}

**Rule 1 (Alternating phases):**

→ Odd and even phases must alternate

**Rule 2 (Reachability):**

→ Each symbol \(S_x\) must reach all symbols \(S0 – S3\) in opposite phase
Step 1: assign arbitrary symbol to arbitrary codeword
**Step 2:** assign symbols to all neighbors of S0 at 0000 in ODD phase

**Rule 1 (Reachability):**
→ Each symbol $S_x$ must reach all symbols $S_0 - S_3$ in opposite phase
1-of-4 LETS Code Derivation: Details

**Step 3:** assign symbols to all neighbors of S1 at 1000 in EVEN phase
Step 3: assign symbols to all neighbors of S1 at 1000 in EVEN phase

S0 already assigned to 0000
**Step 3:** assign symbols to all neighbors of S1 at 1000 in EVEN phase
**Final steps:** complete symbol assignment

Follow same reasoning in previous steps
1-of-4 LETS Code Derivation: Summary

- Codewords in even phase
- Codewords in odd phase

Entire code space filled up

Code space divided into EVEN and ODD phases
1-of-4 LETS Codes: Code Space

- **Many valid 1-of-4 codes possible**
  - 1152 unique codes derivable from method shown
    → Complete enumeration derived in paper

- **Some codes more “practical” than others**
  - All data values easily extracted from codeword

- **Our focus: Two “Practical” codes**
  - “Quasi-1-hot/cold”
  - “Quasi-binary”
### A Practical 1-of-4 LETS Code: “Quasi-1-Hot/Cold”

<table>
<thead>
<tr>
<th>symbol</th>
<th>r3</th>
<th>r2</th>
<th>r1</th>
<th>r0</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>S3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>symbol</th>
<th>r3</th>
<th>r2</th>
<th>r1</th>
<th>r0</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0'</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S1'</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S2'</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>S3'</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>symbol</th>
<th>r3</th>
<th>r2</th>
<th>r1</th>
<th>r0</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>S3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>symbol</th>
<th>r3</th>
<th>r2</th>
<th>r1</th>
<th>r0</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0'</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S1'</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S2'</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>S3'</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

16 codewords for 4 symbols
A Practical 1-of-4 LETS Code: “Quasi-1-Hot/Cold”

<table>
<thead>
<tr>
<th>symbol</th>
<th>r3</th>
<th>r2</th>
<th>r1</th>
<th>r0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>S3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

| EVEN   |    |    |    |    |
| S0     | 1  | 1  | 1  | 1  |
| S1     | 0  | 0  | 1  | 1  |
| S2     | 0  | 1  | 0  | 1  |
| S3     | 0  | 1  | 1  | 0  |

<table>
<thead>
<tr>
<th>symbol</th>
<th>r3</th>
<th>r2</th>
<th>r1</th>
<th>r0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S0’</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S1’</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S2’</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>S3’</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

| EVEN   |    |    |    |    |
| S0’    | 0  | 0  | 0  | 0  |
| S1’    | 1  | 1  | 0  | 0  |
| S2’    | 1  | 0  | 1  | 0  |
| S3’    | 1  | 0  | 0  | 1  |

Code space divided into ODD and EVEN phases
A Practical 1-of-4 LETS Code: “Quasi-1-Hot/Cold”

<table>
<thead>
<tr>
<th>symbol</th>
<th>r3</th>
<th>r2</th>
<th>r1</th>
<th>r0</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>S3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>symbol</th>
<th>r3</th>
<th>r2</th>
<th>r1</th>
<th>r0</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0'</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S1'</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S2'</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>S3'</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Odd code-words**

<table>
<thead>
<tr>
<th>symbol</th>
<th>r3</th>
<th>r2</th>
<th>r1</th>
<th>r0</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>S3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Even code-words**

<table>
<thead>
<tr>
<th>symbol</th>
<th>r3</th>
<th>r2</th>
<th>r1</th>
<th>r0</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0'</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S1'</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S2'</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>S3'</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Multicode:** 2 codewords for each symbol in each phase
A Practical 1-of-4 LETS Code: “Quasi-1-Hot/Cold”

<table>
<thead>
<tr>
<th>Symbol</th>
<th>r3</th>
<th>r2</th>
<th>r1</th>
<th>r0</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>S3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

1-hot

<table>
<thead>
<tr>
<th>Symbol</th>
<th>r3</th>
<th>r2</th>
<th>r1</th>
<th>r0</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0'</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S1'</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S2'</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>S3'</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

1-cold

Quasi-1-hot/1-cold data value easily extracted from codeword
Outline

- Introduction
- Background
- 1-of-4 LETS codes
- 1-of-N LETS codes
- Hardware support
- Analytical evaluation
- Conclusions
1-of-N LETS Codes

► **Goal**
  - To extend solution for 1-of-4 LETS codes to 1-of-N

► **Challenge:**
  - Solution is not obvious for arbitrary N
  - Must satisfy several properties
    → Level-encoding: data can be extracted directly from codeword
    → Transition signaling: each symbol must reach all others via 1 flip
      → alternating phase

► **Contributions**
  - Proof: existence of legal LETS codes for every $N = 2^n$
  - Systematic procedure to generate LETS codes
    → LETS properties formulated as set of constraints
    → Constraints captured in code generator matrix
    → Many different LETS codes exist for each $N$

**See paper for details**
Outline

- Introduction
- Background
- 1-of-4 LETS codes
- 1-of-N LETS codes
- **Hardware support**
  - *Conversion circuit: interfacing channels to nodes*
  - *LETs pipeline circuit: improving channel throughput*
- Analytical evaluation
- Conclusions
LETS Hardware Support: Protocol Conversion

First, focus on protocol conversion circuits

Computation node
Asynchronous
4-phase RZ

Data encode or decode

Asynchronous communication channel (LETS)

Data encode or decode

Computation node
Asynchronous
4-phase RZ
LEDR Converter: Prior Architecture Overview

LEDR Converter from Mitra et al., "Efficient Asynchronous Protocol Converters for Two-Phase Delay-Insensitive Global Communication", ASYNC’07
LED Converter: Prior Architecture Overview

2-phase completion detector

2-phase completion detector

2-phase parity

2-phase parity

2-phase encode

2-phase decode

Four phase function block

Control logic

2/4-phase conversion circuit

2-phase completion detector

2-phase communication channel

2/4-phase conversion circuit

2-phase communication channel
LEDR Converter: Control Signals

- four phase signals
- two phase signals

Diagram showing the flow of signals through blocks labeled LEDR, CD, phase encode, phase decode, control logic, and parity. Arrows indicate the direction of data flow between these components.
New contribution: 1-of-4 LETS Converter

Based on existing LEDR (1-of-2 LETS) converter

- Only minor modifications needed
  - Same overall architecture
  - Most pieces identical
  - Internal logic of some blocks have minimal changes
1-of-4 LETS Converter

= Changed logic blocks
Completion Detector: LEDR vs. 1-of-4 LETS

One layer of C-elements replaced by XNOR gates
**Left Encoder: LEDR vs. 1-of-4 LETS**

![Diagram of LEDR and LETS left encoders]

**Extra layer of XNOR gates**
- Not on critical path!

**LEDRI**
- Data bit: b0, b1
- Enable
- 4-phase true rail: b0, b1
- 4-phase false rail: b0, b1

**LETS**
- Data: r0, r1, r2
- Enable
- 4-phase true rail: b0, b1
- 4-phase false rail: b0, b1

**Note:**
- LEDR vs. 1-of-4 LETS
- Extra layer of XNOR gates
- Not on critical path!
Right Encoder: LEDR vs. 1-of-4 LETS

Extra storage logic
► Not on critical path!

LED right encoder

1-of-4 LETS right encoder
1-of-4 LETS Converter Performance Evaluation

- **Layout performed for LEDR (1-of-2 LETS) conversion circuits**
  Mitra et al., "Efficient Asynchronous Protocol Converters for Two-Phase Delay-Insensitive Global Communication", ASYNC'07
  - With a 4-phase multiplier function block
  - 0.18μm TSMC CMOS process
  - Summary of simulation results:

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward latency</td>
<td>input arrival → output data available</td>
<td>6.8ns</td>
</tr>
<tr>
<td>Stabilization time</td>
<td>input arrival → reset complete</td>
<td>10.5ns</td>
</tr>
<tr>
<td>Pipelined cycle time</td>
<td>min processing time / data item (steady state)</td>
<td>8.3ns</td>
</tr>
</tbody>
</table>

- **1-of-4 LETS expected to add 15 - 20% overhead**
- **Design is delay-insensitive**
  - Except for two simple one-sided timing constraints
LETS Hardware Support: Pipelining Channels

Completed: hardware for \textit{interfacing} with computation nodes
Now focus on: improving performance of global communication through pipelining

Completed: hardware for interfacing with computation nodes
LETS Pipeline: Improving Channel Throughput

**Support #1: MOUSETRAP-based design**
Singh & Nowick, “MOUSETRAP: High-Speed Transition Signaling Asynchronous Pipelines”, TVLSI’07

- **Original MOUSETRAP pipeline**
  - High-speed pipeline scheme for *bundled-data* encoding

- **Proposed design**
  - Pipelines DI communication channel based on MOUSETRAP
  - Eliminates MOUSETRAP bundled-data timing requirements
  - only retains one simple 1-sided timing constraint

- **Simple hardware design**

**Support #2: LEDR-based design**

- **Timing-robust approach, see paper for details**
1-of-4 LETS Pipeline: MOUSETRAP-based design
1-of-4 LETS Pipeline: MOUSTRAP-based design

Latch control: → same as MOUSTRAP

Completion detector: → replaced with 1-of-4 LETS CD
Outline

- Introduction
- Background
- 1-of-4 LETS codes
- 1-of-N LETS codes
- Hardware support
- **Analytical evaluation**
  - *Coding efficiency and transition power metric*
- Conclusions
Analytical Evaluation: Coding Efficiency (LETS vs. RZ)

Coding Efficiency
1-of-N LETS vs. 1-of-N RZ

1-of-N LETS vs. RZ codes
- Same coding efficiency

bits/rails

# of Rails

2           4          8       16      32       64      128    264
Analytical Evaluation: Coding Efficiency (LETS vs. RZ)

1-of-N LETS vs. 1-of N RZ

Coding Efficiency drops off after N>4

1-of-N LETS vs. RZ codes
- Same coding efficiency
Analytical Evaluation: Transition Power (LETS vs. RZ)

Transition Power
1-of N LETS vs. 1-of-N RZ

1-of-N LETS vs. RZ codes
- LETS uses less power

wire-flips/transaction

# of Rails

<table>
<thead>
<tr>
<th># of Rails</th>
<th>LETS</th>
<th>RZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1/2</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1/4</td>
<td>1/2</td>
</tr>
<tr>
<td>16</td>
<td>1/8</td>
<td>1/4</td>
</tr>
<tr>
<td>32</td>
<td>1/16</td>
<td>1/8</td>
</tr>
<tr>
<td>64</td>
<td>1/32</td>
<td>1/16</td>
</tr>
<tr>
<td>128</td>
<td>1/256</td>
<td>1/32</td>
</tr>
<tr>
<td>256</td>
<td>1/512</td>
<td>1/256</td>
</tr>
</tbody>
</table>
Analytical Evaluation: Interpreting LETS Scaling

1-of-N LETS
Transition Power and Coding Efficiency

- **wire-flips/transaction**
- **bits/rails**

# of Rails: 2, 4, 8, 16, 32, 64, 128, 264

Transition Power
Coding Efficiency
Analytical Evaluation: Interpreting LETS Scaling

Trend: Power decreases as # of rails increase but coding efficiency also decreases
Transition Power and Coding Efficiency

Trend: Power decreases as # of rails increase → but coding efficiency also decreases

Sweet spot: going from LEDR to 1-of-4 LETS → halves the power, same coding efficiency
Analytical Evaluation: LETS vs. Synchronous

► **Coding efficiency (# bits encoded/wire)**
  - **Synchronous better than 1-of-N LETS**
    → Synchronous: N bits for N wires
    → 1-of-N LETS: log N bits for N wires

► **Transition power metric (# transitions/wire/data transaction)**
  - **1-of-N LETS better than synchronous as N increases**
    → Synchronous: constant
      → assumes equal probability of wire transition
    → 1-of-N LETS: decreases as N grows
      → $= 1 / \log N$
    → Transition power metric same for $N = 4$
Conclusions

► **A new class of delay-insensitive codes**
  “Level-Encoded Transition Signaling (LETS)”
  - High throughput, low power for global communication
  - Two example 1-of-4 LETS codes shown
  - Generalization to 1-of-N LETS
    → first 1-of-N level-encoded transition signaling scheme

► **Efficient hardware**
  - For protocol conversion to/from four-phase dual-rail signaling
  - For pipelining global communication channel

► **Power and throughput improvements over existing codes**
  - Demonstrated via analytical evaluation
Future Work

- Better evaluation of performance/power metrics
  - Layout of proposed circuits
  - Evaluation of second-order effects
    → e.g. cross-coupling, noise, etc

- Extend conversion circuits to support other encoding styles
  - e.g. 1-of-4 RZ, single-rail bundled
Appendix
**LEDR Converter: System Simulation**

**Step 1: Two-phase inputs arrive**

*LEDR inputs begin arriving at quiescent system*

![Diagram showing the LEDR converter system simulation](image-url)
**LEDR Converter: System Simulation**

**Step 2: Two-to-four phase conversion**

*Input completion detection sent to control*

- **LEDPR** Converter
  - **Phase signal changes**
  - **control logic**
  - **four phase encode**
  - **four phase function block**
  - **four phase decode**

Diagram showing flow of data and control signals between LEDR input and output blocks.
**Step 2: Two-to-four phase conversion**

Control enables four-phase evaluate phase
Step 2: Two-to-four phase conversion

LED Converter: System Simulation

Enable now high

LED input converted to four-phase

LEDR input converted to four-phase

LED input

LED output

Phase

Enb

Comp

Data

Parity
**Step 3: Four-phase evaluate**

**Four-phase function evaluation**

Diagram showing the LEDR Converter: System Simulation with four-phase encode, four phase function block, four phase decode, control logic, LEDR input, LEDR output, data, parity, phase, enb, comp, ack_left, and ack_right.
Step 4: Four-to-two phase conversion

Four-phase bits decoded to LEDR
Step 4: Four-to-two phase conversion

Ack from right may arrive at any time after all pairs are sent
**Step 5: Four-phase reset**

*Control enables four-phase reset phase*

**LED Converter: System Simulation**

- Data parity LEDR input
- LEDR output parity
- Control logic
- Enable falls
- Phase
- Ack_left
- Ack_right
Step 5: Four-phase reset

Function block inputs return to zero

Pipeline concurrency: Request new data during reset
Step 5: Four-phase reset

Four-phase reset propagates through logic block

Complete falls
**LEDR Converter: System Simulation**

**Ready to evaluate again**

*New evaluate phase begins when enable rises again*

![Diagram of LEDR Converter](image)

- **LEDR input**
  - LEDR
  - CD
  - parity
  - phase

- **LEDR output**
  - LEDR
  - CD
  - parity

- **Dataflow**
  - (Data) → **LEDR input** → **four phase encode** → **four phase function block** → **four phase decode** → **control logic** → **LEDR output**

- **Control Signals**
  - enb
  - comp
  - phase
  - ack_left
  - ack_right