iNc Infinite Storage System Whitepaper

In-Memory Computing: Redefining Data Storage

LC-INC-003 1.0

Core concept

Core concept

Transform storage devices from a "centralized control (Controller + Passive NAND)" topology to a "decentralized/distributed computing network (Mesh Network of Computing NANDs)".

Utilizing the stateless characteristics of RLNC (Random Linear Network Coding), combined with in-Nand computing (iNc) capabilities, it is theoretically possible to break the physical limitations of traditional SSD master control CE (Chip Enable) and channel number, and achieve an infinite expansion architecture similar to the "Storage Internet".

If this architecture is successfully implemented, Lingcun will have an advantage that competing products cannot match: system-level robustness.

This solution breaks out of the limitations of the physical layer and uses mathematics (RLNC) and logic (INC + MRD) to build a storage life that theoretically never loses data and is physically infinitely expandable.

This is the most cutting-edge intersection in the global storage field. If Lingcun can take the lead in running this "master-less self-balancing storage system based on RLNC" on FPGA, it will redefine what a "hard drive" is.

1. Feasibility analysis

Technical logic analysis

The technical logic is established. The difficulty in hardware implementation is not storage, but the interconnect topology.

- Traditional SSD: The main control is the "brain", and the NAND is the "warehouse". If the warehouse is too big, the brain can't handle it (CE limitation).

- Current idea: Each NAND becomes an "intelligent node", which counts as existing. Data is no longer stored at a fixed point, but flows and resides in the network in the form of RLNC coded slices.

Key challenges: Standard NAND Flash (even with in-memory computing capabilities) is physically a slave device (Slave Device). It has no clock source and cannot actively initiate transfers (Master).

- The essence of NAND: Standard NAND is a Slave (slave device). It has no "autonomous consciousness" and will only act when it receives instructions (Read/Program). It does not know "who I am" or "who is my neighbor".

- Requirements for the RLNC network: Each node must be an Autonomous Agent.

- It needs to actively initiate a handshake.

- It needs to judge TTL (time to live).

- It needs to manage Credit (flow control).

- It needs to actively "push" data packets to neighbors (Forwarding).

Pure storage array circuits cannot achieve these "active behaviors". There must be a logic core (finite state machine FSM or microcontroller) to give it "intelligence", this brain must be on the Logic Die.

High-speed interconnect physical layer (SerDes PHY)

Since chip-to-chip interconnect requires high-speed SerDes (differential serial interface).

- Physical Reality: SerDes circuit contains PLL (Phase Locked Loop), CDR (Clock Recovery), Equalizer (Equalizer). These are analog mixed signal circuits.

- Limitations of NAND: The manufacturing process of NAND Flash makes it very difficult to create high-performance analog circuits. If you forcibly build 16Gbps SerDes on NAND Die, the signal jitter will be so large that communication will be impossible.

- Logic Die Advantages: Standard CMOS processes are very good at doing SerDes PHY.

In order to realize the "highway" between chips, the physical interface must be provided by Logic Die.

Computing energy efficiency

Although it is possible to implement simple bit operations (such as XOR) by modifying the Page Buffer's latch logic, this is a "best effort" calculation.

- Requirements for RLNC: Involves GF(2^8) finite field multiplication, which requires a large number of look-up tables (LUTs) or shift XOR operations.

- Efficiency comparison:

- Doing it next to a NAND array: it can take hundreds of microseconds (us) to complete a matrix operation because the transistors can't run fast.

- Do it on Logic Die: only takes a few nanoseconds (ns).

Without Logic Die, the read and write speeds of the "Infinite Hard Drive" may be as slow as a floppy disk, unable to meet the needs of AI or data centers.

Smart NAND: intelligent stacking

Smart NAND (intelligent stacking) = Storage Die (pure storage array) + Bonded Logic Die (in-memory computing + routing logic)

In this form:

1. No need for a large controller (Main Controller) at the PCB board level in the traditional sense.

2. No additional CPU or DRAM chips required.

3. But it is required to bond a thin CMOS logic layer through 3D packaging on the bottom (or top) of each NAND particle.

Therefore, Logic Die still exists, but it has changed from "a large chip on the motherboard" to an "invisible interlayer in the NAND particle".

This is also the key to reducing costs:

- Storage wafers focus on storage (higher yield, lower cost).

- Logic wafers focus on computing (using mature 40nm process, extremely small area, extremely cheap).

- Once the two are combined, they form an "infinitely scalable node with computing and routing capabilities"

Solution: It is necessary to add a minimalist micro routing unit (Micro-Controller/Router) next to each NAND or each group of NAND (such as a Die Stack), or use in-memory computing technology to give the NAND some form of "bus driver" capability (this is extremely challenging and usually requires the cooperation of an external FPGA/ASIC).

2. Detailed technical approach

Can storage capacity expand indefinitely?

Yes, but the physical form will change.

- Logic: With RLNC, the data is split into n encoded packets. The data exists as long as there are enough nodes in the network to store these packets. To increase capacity, simply "connect a new node", and the new node automatically participates in calculation and recoding, without the need for a central master to remap the logical address (LBA).

- Implementation: This is no longer a standard SSD, but a Hyper-scale Array. Its scalability is similar to the Internet. As long as nodes (NAND + micro control logic) are added, the capacity and computing power will increase linearly.

Cooperative MRD and LDPC error correction

This is a great combination, layered governance.

- Bottom layer (physical layer): LDPC. For the bit flip and oxide layer wear of the NAND Flash medium itself. This is "road building" within NAND.

- Upper layer (system layer): MRD (maximum rank distance code). MRD outperforms traditional RS codes in handling errors in network coding systems (such as rank loss, malicious node contamination, packet loss).

Workflow: In-memory calculation first uses LDPC to calibrate the read data -> RLNC/MRD calculation layer processes the linear combination of data packets -> output/forward. If a NAND particle is completely damaged (node failure), MRD combined with RLNC features can recover data by simply collecting encoding packets of sufficient dimensions from surrounding nodes without knowing which location is broken.

Can NAND Flash replace routers for forwarding?

Technical Difficulty: As mentioned before, standard NAND is a Slave.

Path:

- Option A (in-memory computing +): If the "memory" technology allows NAND to generate signals on the I/O bus, it can be achieved.

- Option B (Associated Logic): A more realistic approach is the NoC (Network on Chip) architecture. A very small logic die is attached next to each NAND particle (or integrated within the package). This logic die is responsible for handshaking, routing and driving the bus, while the NAND core is only responsible for storage and matrix calculations.

Forwarding logic: The data packet comes in -> the NAND computing unit determines whether it belongs to itself -> if not, it uses the in-memory computing logic gate to transparently transmit it directly to the next port.

Controller design

Centralized main control disappears and evolves into "Gateway" + "distributed micro-control".

- Pre-development: FPGA must be used because the encoding and decoding logic of RLNC and the

communication protocol between NAND need to be verified.

- Final form:

- Interface side (Gateway): Processes the NVMe protocol and pretends to be a standard hard drive to the host. It is only responsible for slicing and "injecting" user data into the NAND network, and is not responsible for managing specific physical addresses.

- Node side: Completely decentralized. Each NAND node runs a set of microcode responsible for: receiving packets, doing RLNC XOR/matrix multiplication, storing or forwarding.

Firmware changes

Completely transform from FTL (Flash Translation Layer) to "routing and encoding protocol stack".

- Traditional FW: Logical address -> physical address mapping, garbage collection, wear leveling.

- New FW (distributed):

- Header parsing: Identifies the RLNC coefficient vector.

- Gossip Protocol: Exchange load information between nodes ("I'm full, don't send it to me").

- Partial repair: If the neighbor node is found to be invalid, reconstruction calculation will be automatically initiated.

Time-to-live (TTL) design

In RLNC network forwarding, TTL refers to "hop limit".

- Design: Prevent packets from looping infinitely in the NAND array.

- Mechanism: Each packet header contains a TTL counter. Each time it passes through a NAND node for calculation/forwarding, the TTL is reduced by 1. When TTL=0 and is not stored, discard the packet and request retransmission from the source (or trigger MRD error correction).

- Storage TTL: For cold data, set the "re-encoding cycle" and use in-memory calculations to read and refresh the encoding coefficients regularly to prevent data loss caused by charge leakage.

Hardware circuit design

Topology: 2D/3D Mesh or Torus structure, not bus structure.

- Traditional SSD: Multiple parallel channels (Channels), each mounting multiple CEs.

- RLNC Array: Similar to FPGA internal interconnect.

- Point-to-point interconnection: Chip A connects Chip B and Chip C.

- High-bandwidth interface: The traditional 8-bit wide ONFI bus may not be enough. It is recommended to use a high-speed serial interface (SerDes) similar to HBF to reduce the number of pins and increase the interconnection density.

Product form: array or standalone SSD?

Go in two steps.

- Form 1 (supercomputing level): Storage array/cabinet, providing Ethernet/Infiniband interface to

the outside, and a large number of NAND nodes internally. Suitable for verifying "infinite expansion".

- Form 2 (consumer grade): A single PCIe card, which looks like a graphics card and is covered with NAND. The external interface is an NVMe interface. It uses the high concurrent throughput capability of RLNC to provide the ultimate IOPS.

HBF chip interconnect planning

If HBF is used, its high-bandwidth characteristics perfectly fit the computational needs of RLNC.

- Circuit planning: Use TSV (through silicon via) stacking technology or interposer (interposer) interconnection.

- Pipeline design: Data flows through the HBF chain like a pipeline, Chip 1 does $A \times K_1$, Chip 2 does $B \times K_2$, and the results are accumulated during transmission.

Data storage and retrieval strategy

Evenly distributed across the entire network (similar to DHT, distributed hash table).

Write (Store): Do not fill linearly (do not fill Chip 1 first and then Chip 2).

- Gateway slices and RLNC encodes the data.

- Utilize multi-path routing to "spray" encoded packets like a fountain throughout the network.

- The front chip acts as a routing node and pushes data to the deep nodes first. After the deep nodes are fully loaded, the shallow nodes are backfilled.

Read: Host requests data -> Gateway issues query -> Any k nodes with relevant encoding packets respond -> Aggregate to Gateway for decoding. Using in-memory computing, partial decoding can be performed on the transmission path to reduce bus traffic.

Implementation logic: Data no longer exists in the form of "files", but in the form of "solutions to mathematical equations".

Writing process:

1. Host writes 4KB data.

2. Gateway is divided into k symbols and generates n random coefficient vectors ($n > k$).

3. Gateway sends (data fragment, coefficient) tuple into the NAND network.

4. The NAND node receives, uses in-memory computing to linearly combine the new data with the old data (if aggregation is required), and writes them to the Flash Page.

Reading process:

1. Gateway collects any k combination packages.

2. Gaussian elimination (at Gateway or accelerated by FPGA) restores the original 4KB.

Garbage collection and wear leveling

Wear Leveling: Natural advantage. RLNC breaks up the data, which itself is a kind of randomized writing. No complex centralized WL algorithm is required. A node simply checks its own PE number

when writing, and if it is too high, new writes are rejected via the routing protocol, forcing data to flow to low-wear nodes.

Garbage Collection: When a file is deleted, Gateway broadcasts the "Expiration Factor ID". The node uses in-memory computing in the background to scan the header and mark invalid blocks. No data transfer is required, just delete it directly.

Preventing data loss

Mathematical guarantee: As long as the number of linearly independent encoding packets in the entire network ≥ the number of original data fragments, the data can be recovered.

Physical Security:

- Heartbeat monitoring: Adjacent nodes monitor each other's survival status through low-power signals.

- Automatic regeneration: Once a node in a certain area is found to be offline, surrounding nodes use in-memory calculations to generate new encoding packages based on existing data to fill in redundancy (without reading the original data, directly re-encoding the encoded data).

Avoiding global stalls during garbage collection

In the decentralized RLNC + in-memory computing (INC) architecture, garbage collection (GC) is no longer a heavy "global synchronization" task, but has evolved into a local self-healing process similar to the metabolism of biological systems.

Since there is no centralized master to maintain the global physical mapping table, it must be solved: when a node is erasing a block (Erase), how to prevent the delay of the entire Mesh network from experiencing violent jitters?

Core innovation: Recoding as GC

In a traditional SSD, the GC needs to move valid pages to new blocks, which results in a large amount of internal read and write amplification. In the new architecture, since the INC node has computing capabilities, we can implement GC through "recoding":

1. Partial compression: When the invalid pages in a Block reach the threshold, the INC logic is not simply moved, but reads all valid encoding packets belonging to different GIDs in the Block.

2. In-memory fusion: Linear Combination of these packets with the newly arrived write stream in the Buffer.

3. One-shot writeback: Generate a completely new encoding packet and write new chunks.

- Advantages: The GC process has become a process of "enhancing data redundancy", and the "reinforcement" of the data is completed while moving the data.

Three mechanisms for avoiding global stalls

1. Credit limit stabilization mechanism (Credit-based GC Throttling)

We mentioned Credit (credit limit) in the communication protocol design.

- Logic: When node A needs to be physically erased (this is the most time-consuming operation of NAND, about 3-5ms), it will actively set the externally announced Credit to 0.

- Effect: Neighbor nodes will automatically bypass this node for routing, or temporarily cache the data packet in its own Buffer. Because it is a Mesh network, the path is multi-dimensional, and the traffic will automatically detour like water meeting a stone, thus avoiding global I/O stalls.

2. "Drop & Reconstruct" strategy (Drop & Reconstruct)

This is the unique "domineering" approach of the RLNC solution:

- Traditional solution: Every valid Page must be preserved during GC, otherwise the data will be lost.

- RLNC solution: If a node has an extremely high load and urgently needs to erase the Block to make room, it can directly "discard" some valid encoding packets in the block, and then issue a RECODE_REQ (reconstruction request).

- Self-healing: After receiving the request, the gateway or neighbor node uses the redundancy feature of RLNC to generate new encoding packets on other idle nodes to make up for the missing rank (Rank).

- Conclusion: Trade space (redundancy) for time (zero latency).

3. Asynchronous Predictive Erase

Use in-memory computing logic to analyze write patterns:

- If the INC logic predicts that there will be large traffic injection in the next 100ms (based on the traffic forecast of the front-end gateway), it will preemptively execute the Erase instruction during the idle period.

- Since 3D NAND has multiple Planes, Independent Plane Operation can be implemented: Plane 0 does GC erasure, and Plane 1 still responds to reading normally without interfering with each other.

Natural wear leveling

Under the "infinite capacity" vision, wear leveling is done automatically:

- Space for life: Since the capacity is infinitely expandable, data is randomly distributed in the network.

- Automatic hotspot drift: If a node writes too much and the number of PEs increases, its processing speed will slow down in microseconds. RLNC's routing protocol will automatically sense this increase in delay, thereby reducing the weight sent to the node.

- Algorithm-Free Algorithm: This is a dynamic balance based on physical feedback that is more efficient than any complex software WL algorithm.

3. Overall design principles

Mapping packets to NAND pages

Physically map packets to NAND Page structures.

The traditional SSD Page structure is usually: User Data (4KB/8KB/16KB) + OOB/Spare (several hundred bytes).

Disruptive Design:

- OOB/Spare area → Routing & Coding Header.

- Reason: In-memory computing circuits can often read OOB areas quickly without having to activate the data transfer of the entire Page. In this way, the node can read the header first to determine whether calculation or forwarding is required, greatly reducing power consumption and latency.

- User Data area → Encoded Payload.

- Reason: Stores data entities after linear combination.

Detailed packet structure

Suppose we define a logical atomic unit (Atomic Unit), corresponding to a physical 4KB Page (plus 128 Bytes OOB as an example).

Physical Layer Container

This layer mainly handles media error correction, which is automatically handled by the underlying hardware.

Region	Field Name	Length	Description
Tail	LDPC Parity	Variable length	Physical error correction code (bit flip protection) against NAND media errors. The underlying circuit is automatically stripped away, and the upper logic is invisible.

A. Routing and control header

Function: Responsible for controlling the flow of data between chips and channels. Located in front of OOB area.

Offset	Field name (Field)	Bit width (Bits)	Detailed definition and purpose
0x00	Packet Type	4	Identifies the packet type: 0001: Original data packet (Systematic), 0010: Coded data packet (Coded), 0100: Control command (Control/GC), 1000: Request retransmission (Request)
0x04	TTL (Time to Live)	4	Hop limit. Decrease by 1 for each NAND node passed. Protect against loop storms.

Offset	Field name (Field)	Bit width (Bits)	Detailed definition and purpose
0x08	Target Node ID	16/32	Target logical area ID. If it is broadcast mode (full network storage), it is a specific mask.
0x20	INC OpCode	8	In-memory calculation instruction. Tell NAND what to do with this packet: 0x01: Forward Only (Route Only), 0x02: XOR & Store, 0x03: Multiply and Accumulate (MAC & Fwd)
0x28	MRD Checksum	32	Maximum rank distance code checksum. For fast detection of contamination of linear subspaces (detection of errors without decoding).

B. RLNC coding header

Function: Describes the mathematical properties of data, used for decoding and recoding. Located in the back section of OOB area.

Offset	Field name (Field)	Bit width (Bits)	Detailed definition and purpose
0x00	Generation ID (GID)	64	Generation/Batch ID. All slices belonging to the same original file (or LBA segment) share this ID. This is the prerequisite for combined calculations.
0x40	Symbol Index	16	If it is an original data packet, it indicates which slice it is. If it is a fully encoded package, this item can be ignored or set to all 1s.
0x50	Coding Vector / Seed	128~256	Core field: Coding coefficient vector. Optimization solution: In order to save OOB space, the complete coefficient vector is not stored directly, but a PRNG Seed (random number seed) and Mask are stored. The sink NAND utilizes INC logic to locally generate the coefficient vector based on the seed.
0xVariable	Current Rank	8	The number of independent equations contained in the current linear combination (used to determine whether enough data has been collected to restore the original information).

C. Payload field

Function: Carrying information entities. Located in the Main Data area.

Field name	Size	Description
Encoded Data Symbols	4KB (typ)	This is a vector $P = \Sigma(c_i \cdot D_i)$. What is stored here is not the original text of

Field name	Size	Description
		the file, but the result of the original text after arithmetic operations in the GF(2^q) field.
Padding	Variable length	If the original data is less than 4KB, zero padding is used to align the NAND Page to facilitate alignment of matrix operations.

Key technical optimizations

In order to maximize performance in "unlimited capacity" and "in-memory computing" scenarios, it is recommended to adopt the following two optimization strategies:

Seed-Based Compression Technology

In RLNC, as the generation size k increases, the coefficient vector becomes very large (for example, k=128, with 1 byte per coefficient, the vector requires 128 bytes, occupying the OOB).

- How to do it: Do not transmit c_1, c_2, ..., c_k in the Header.

- Improvement: Only transfer one Random Seed (32bit).

- Calculation: The NAND node's INC circuit has a fixed pseudo-random number generator (such as LFSR) built into it.

- Coefficients = LFSR(Seed)

- Benefits: The header size is greatly reduced, leaving more space for routing control and MRD verification.

Sparse Network Coding (Sparse RLNC)

To reduce the power consumption of NAND in-memory computation, do not have each packet be composed of all k original packets.

- Design: Limit the density of encoding vectors.

- Header implementation: Add Density Map field in Header.

- Benefits: Reduces the number of floating point/integer multiplications during in-memory calculations and improves throughput.

Workflow: a packet traversing NAND devices

Assuming that NAND-A receives a data packet, the process is as follows:

1. Header sniffing (1-3 clock cycles):

- The interface logic of NAND-A only reads the OOB area.

- Check the Target Node ID → found that it is sent to this area.

- Check TTL → TTL > 0, valid.

- Check MRD Checksum → the packet is not contaminated.

In-memory judgment (INC Logic):

- Read the Generation ID (GID).

- INC logical search local storage array: "Do I have other data shards for this GID?"

- Case 1 (Storage): If the local area is empty, write the Payload directly to the Flash unit.

- Case 2 (calculation aggregation): If there is data for the GID locally, start in-memory matrix addition:

- New_Payload = Payload_in ⊕ Payload_stored

- New_Vector = Vector_in ⊕ Vector_stored

- Case 3 (forwarding): If the local area is full, modify the Next Hop in the Header and push it to NAND-B using the bus.

Execute action:

- The entire process does not require the intervention of the main control CPU, and is all completed in the NAND edge logic and Cell array.

For this product, the design of the Header determines the upper limit of the architecture.

- Suggestion 1: Use the design of "fixed-length Header + variable-length Vector" to make use of OOB space.

- Recommendation 2: INC OpCode is the core innovation that turns NAND from memory into an instruction executor. You need to define a simplified instruction set (ISA) embedded in the Header.

"iNc OpCode (in-memory computing instruction set)" is related to the specific logic design of the hardware circuit. This is a very critical link. Transforming NAND Flash from "passive storage" to "active computing node" requires a streamlined, efficient and dedicated instruction set for RLNC operations. This instruction set does not need to be as general as x86 or ARM. It is more like an ASIP (Application-Specific Instruction-set Processor) optimized for linear algebra and data flow.

4. INC OpCode instruction set

INC OpCode instruction set overview

OpCode is divided into three major categories: Flow, Compute, and Maintenance.

Recommended bit width: 8-bit (supports 256 instructions, currently defines about 10 core instructions, leaving a lot of expansion space)

Flow-control instructions

Used for data transportation and routing without modifying data content.

OpCode (Hex)	Mnemonic	Full name of instruction	Hardware behavior logic
0x00	NOP	No Operation	No operation. Used to align clocks or wait for bus arbitration.
0x01	PASS	Pass Through	Route forwarding. Without reading the Flash array, the data from the input port is directly transferred to the output port (next hop). Triggered when TTL > 0 and local storage is full.
0x02	STORE	Direct Store	Direct storage. Write the payload directly to the Flash Page without calculation. Used to store system codes (copies of original data).
0x03	BCST	Broadcast	Broadcast. Copies and sends the data packet to all neighbor nodes, often used to distribute global configuration or urgent error correction requests.

Compute and RLNC instructions

This is the core of in-memory computing, performing mathematical operations on the GF(2^q) domain.

OpCode (Hex)	Mnemonic	Full name of instruction	Hardware behavior logic	Mathematical expression
0x10	XOR_WR	XOR & Write	Read the target Page, perform XOR with the input data, and write the result back. This is the most basic encoding operation of RLNC.	$D_{new} = D_{in} \oplus D_{old}$
0x11	MAC_WR	Multiply-Accumulate & Write	Core instructions. The input data is multiplied by the coefficient α, the stored data is multiplied by the coefficient β, and the result is written back.	$D_{new} = \alpha \cdot D_{in} \oplus \beta \cdot D_{stored}$

OpCode (Hex)	Mnemonic	Full name of instruction	Hardware behavior logic	Mathematical expression
0x12	RECODE	Regenerate Code	Regenerate code. Does not rely on external input. Read local k Pages, generate a new random linear combination packet and transmit it. Used to repair lost data in the network.	$P_out = \Sigma r_i \cdot P_stored_i$
0x13	CHK_MRD	Check MRD Syndrome	Integrity check. Calculate the syndrome of the data to determine whether there is rank loss or bit flipping. If the verification fails, RECODE is automatically triggered.	S = H · D^T

Maintenance instructions

Used for garbage collection and state management.

OpCode (Hex)	Mnemonic	Full name of instruction	Hardware behavior logic
0x20	INV_GID	Invalidate GID	Mark data for a specific Generation ID as "invalid/garbage". The physical block is not erased, only the metadata area is marked, waiting for background GC.
0x21	RPT_STAT	Report Status	Report its own status (wear PE Count, remaining capacity, current load) to the gateway or neighbor.

Instruction encoding

In order for NAND's INC logic to be parsed quickly, the instruction format should be fixed and streamlined.

Recommended format (1 Byte OpCode + optional operands):

bitfield	name	description
[7:5]	Category	Instruction category: 000=Flow, 001=Compute, 010=Maint
[4:0]	Sub-Op	Sub-Opcode (32 seed instructions)

Operands source:

- Implicit operands: Most operands come from other fields in the Header (such as GID, Coding Vector) and do not need to be appended after the OpCode.

- Explicit operand: Only when a specific Page address needs to be specified (such as RECODE specifying the source Page), the address of 2-4 Bytes is appended to the OpCode.

Hardware implementation recommendations

1. Micro ALU (Arithmetic Logic Unit):

Implement an ALU optimized for GF(2^8) on Logic Die.

- Core unit: 8-bit XOR array + look-up table multiplier (LUT-based Multiplier).

- Area: Very small, about 0.01 mm^2 (40nm process).

2. Instruction Decoder:

A simple combinational logic circuit reads OpCode bytes and outputs control signals to the ALU and Flash interfaces.

3. State Machine (FSM):

Since instructions are executed in a streaming manner (packet arrival → decoding → execution → emission), no complex pipeline is required. A 3-5 state FSM is sufficient: IDLE → FETCH_HEADER → DECODE → EXECUTE → EMIT.

5. Hardware circuit design

Hardware circuit design overview

In order to realize the vision of "infinite capacity memory", a new hardware architecture needs to be designed. The core of this architecture is to transform the traditional "master + passive NAND" model into a "Mesh network composed of intelligent NAND nodes."

Smart NAND node architecture

Each Smart NAND node consists of two parts:

1. Storage Die (storage layer):

- Standard 3D NAND Flash array

- Responsible for the physical storage of data

- Provide LDPC error correction capabilities

2. Logic Die:

- Bonding with Storage Die via 3D packaging

- Contains INC compute unit, routing logic, SerDes PHY

- Responsible for data calculation, forwarding and protocol processing

Interconnection method:

- Using TSV (Through-Silicon Via) or Hybrid Bonding technology

- The bandwidth between Logic Die and Storage Die can reach TB/s level

Logic Die core modules

A. INC Compute Engine

Modules	Functions	Area Estimation
GF(2^8) ALU	Finite field multiplication and addition	0.01 mm²
XOR Array	Batch XOR operation	0.005 mm²
LFSR	Pseudo-random number generation	0.001 mm²
Instruction Decoder	OpCode Analysis	0.002 mm²

B. Routing Engine

Modules	Functions	Area Estimation
Header Parser	Parse OOB area	0.003 mm²
Routing Table	Neighbor node address table	0.01 mm²
Credit Manager	Flow Control Management	0.005 mm²
TTL Processor	Hop count management	0.001 mm²

C. Physical Interface

Modules	Functions	Area Estimation
SerDes PHY	High-speed serial interface	0.1 mm²
PLL	Clock generation	0.02 mm²
CDR	Clock Recovery	0.03 mm²

Mesh network topology

Topology selection: 2D Torus

- Each node is connected to 4 neighbors (top, bottom, left, right)

- Boundary nodes are connected around each other to form a torus

- Advantages: path redundancy, load balancing, no single point of failure

Physical layout example (4x4 array):

[N00]--[N01]--[N02]--[N03]--[N00]

| | | | |

[N10]--[N11]--[N12]--[N13]--[N10]

| | | | |

[N20]--[N21]--[N22]--[N23]--[N20]

| | | | |

[N30]--[N31]--[N32]--[N33]--[N30]

| | | | |

[N00]--[N01]--[N02]--[N03]--[N00]

Extension method:

- Horizontal expansion: increase the number of columns

- Vertical expansion: increase the number of rows

- 3D extension: stack multiple layers of 2D Torus

Gateway design

Gateway is the interface between the entire system and the external world, responsible for protocol conversion and data injection/extraction.

Core features:

1. NVMe protocol processing

- Parse NVMe commands sent from the host

- Convert logical address to Generation ID

2. RLNC Encoder

- Split user data into k symbols

- Generate n encoding packets (n > k)

- Assign a vector of random coefficients

3. RLNC decoder

- Collect k linearly independent coding packets

- Gaussian elimination method restores original data

4. Traffic Scheduler

- Distribute data flow according to network load

- Implement load balancing

Hardware implementation:

- It is recommended to use FPGA for prototype verification

- Mass production version can be integrated as ASIC

6. Physical-layer protocol

Physical-layer protocol overview

In order to achieve efficient communication between Smart NAND nodes, a dedicated physical layer protocol needs to be designed. This agreement needs to meet the following requirements:

- High Bandwidth: Supports 16Gbps+ data transfer rate

- Low Latency: End-to-end latency < 100ns

- Reliability: Bit Error Rate < 10^-15

- Extensible: Supports dynamically increasing and decreasing nodes

Frame format

Physical frame structure:

Field	Length	Description
Preamble	8 Bytes	Synchronization sequence for clock recovery
SFD	1 Byte	Frame start delimiter
Header	16 Bytes	Routing and encoding header
Payload	4KB	Data payload
CRC	4 Bytes	Frame check sequence
EFD	1 Byte	End of frame delimiter

Transmission efficiency: Payload ratio > 99%

Flow-control mechanism

Adopt a credit-based flow control mechanism:

1. Initialization phase:

- Each node announces its buffer depth to its neighbors

- Neighbors record the node’s initial Credit

2. Data transmission:

- Sending a data packet consumes 1 Credit

- After the receiver completes processing, return to Credit update

3. Congestion handling:

- When Credit = 0, suspend sending to this neighbor

- Packets are automatically routed to other neighbors

Advantages: No packet loss, no retransmission, predictable delay

Handshake protocol

Node joining process:

1. Discovery phase: The new node broadcasts the HELLO message

2. Authentication phase: Neighbor nodes verify the identity of the new node

3. Configuration phase: Assign node ID and synchronize routing table

4. Ready phase: New nodes begin to participate in data storage and forwarding

Node exit process:

1. Notification phase: The node broadcasts the GOODBYE message

2. Migration phase: Neighbor nodes trigger data recoding

3. Cleaning phase: Update the routing table and remove the node

7. Dynamic expansion

Dynamic expansion overview

The core of "unlimited capacity" is that the system can dynamically add storage nodes at runtime without downtime or data migration. This is the fundamental advantage of RLNC architecture over traditional RAID.

Hot-add node procedure

1. Physical connection:

- New nodes connect to the Mesh network

- Automatically discover neighbor nodes

2. Capacity registration:

- New nodes report their capacity to the Gateway

- Gateway updates global capacity view

3. Data balancing:

- The system automatically migrates some encoding packages to new nodes

- Using the RLNC feature, you only need to generate a new encoding package without moving the original data

4. Completion of expansion:

- The new node begins to participate in reading and writing normally

- System capacity increases linearly

Key advantages: The entire process is transparent to upper-layer applications and requires no downtime

Hot-remove node procedure

1. Mark for retirement:

- The administrator marks a node as being about to be retired

- The system stops writing new data to the node

2. Data recoding:

- The neighbor node reads the encoded packet of the node

- Generate new encoding packages and store them in other nodes

- Ensure redundancy is not reduced

3. Safe removal:

- Confirm that the data has been fully re-encoded

- Physically disconnect nodes

Fault Tolerance: Even if a node suddenly fails (untagged), the system can automatically recover

Capacity calculation

Effective capacity = Total physical capacity × (k/n)

Among them:

- k = number of original data shards

- n = total number of encoded packets (n > k)

- (n-k)/n = redundancy overhead

Example:

- Physical capacity: 100TB (100 1TB nodes)

- k=8, n=12 (1.5x redundancy)

- Effective capacity = 100TB × (8/12) = 66.7TB

After expansion:

- Physical capacity: 150TB (50 new nodes added)

- Effective capacity = 150TB × (8/12) = 100TB

- Capacity grows linearly without data reorganization

8. Applications of MRD codes

MRD codes overview

MRD (Maximum Rank Distance) code is an error correction code specially designed for network coding systems. Unlike traditional RS codes, MRD codes can efficiently detect and correct "rank-deficient" type errors.

MRD and Reed–Solomon codes compared

Features	RS code	MRD code
Error model	Sign error	Rank loss
Applicable scenarios	Traditional storage	Network coding
Detection capabilities	Locating error locations	Detecting subspace contamination
Error correction ability	Correct t symbols	Correct t-dimensional rank loss
Computational complexity	$O(n^2)$	$O(n^2)$

Key advantages: MRD code does not need to know which node is faulty, it only needs to detect whether the overall rank is sufficient

MRD in infinite storage

1. Data integrity check:

- Each encoded packet carries MRD checksum

- Nodes can quickly detect whether data is contaminated

- Determine data validity without decoding

2. Malicious node detection:

- If a node returns incorrect data

- MRD verification will detect abnormalities immediately

- The system automatically isolates the node

3. Efficient error correction:

- When rank loss is detected

- Just get extra encoding packages from other nodes

- No need to retransmit all data

4. In collaboration with LDPC:

- LDPC handles physical layer flipping

- MRD handles system level rank loss

-Double protection, data will never be lost

MRD hardware implementation

Calculation unit:

- Matrix operations based on GF(2^8)

- Share ALU resources with RLNC

Verification process:

1. Read the coefficient vector of the encoding packet

2. Calculate the adjoint formula $S = H \times V^T$

3. If $S \neq 0$, trigger the error correction process

Area Overhead:

- Extra logic < 0.01 mm^2

- Reusable RLNC computing unit

9. INC-Core module inventory

INC-Core module overview

INC-Core is the core logic unit of a Smart NAND node, integrating all necessary computing, routing and management functions. Below is the complete list of modules.

Compute modules

Module name	Function description	Interface	Area
GF_ALU	GF(2^8) Arithmetic Logic Unit	8-bit in/out	0.01 mm²
XOR_ARRAY	4KB batch XOR	4KB in × 2, 4KB out	0.005 mm²
LFSR_GEN	Pseudo-random coefficient generation	32-bit seed, 8-bit out	0.001 mm²
MATRIX_MUL	Matrix Multiplication Accelerator	Configuration Interface + Data Flow	0.02 mm²
GAUSS_ELIM	Gaussian elimination unit	Matrix input, solution vector output	0.03 mm²

Routing modules

Module name	Function description	Interface	Area
HDR_PARSER	Header parser	OOB input, control signal output	0.003 mm²
ROUTE_TABLE	Routing table storage	Address query interface	0.01 mm²
CREDIT_MGR	Flow control credit management	Neighbor status interface	0.005 mm²
TTL_PROC	TTL processor	Counter interface	0.001 mm²
PKT_SWITCH	Packet Switch	4-port I/O	0.015 mm²

Interface modules

Module name	Function description	Specifications	Area
SERDES_PHY	High-speed serial transceiver	16Gbps × 4 lanes	0.1 mm²
PLL	Phase Locked Loop	100MHz - 8GHz	0.02 mm²
CDR	Clock Data Recovery	Adaptive Equalization	0.03 mm²
FLASH_IF	Flash Array Interface	ONFI 4.0 Compatible	0.01 mm²

Control modules

Module name	Function description	Interface	Area
FSM_CTRL	Main state machine	Global control signal	0.005 mm²

Module name	Function description	Interface	Area
INSTR_DEC	Instruction decoder	OpCode input, control output	0.002 mm²
INT_CTRL	Interrupt controller	Exception handling interface	0.002 mm²
PWR_MGR	Power Management	Sleep/Wake Control	0.003 mm²

Total area estimate: < 0.3 mm² (40nm process)

Power consumption estimate: < 50mW (typical working state)

10. Applications and advantages

Application overview

The unlimited capacity storage system based on RLNC + INC has revolutionary application value in many fields.

AI data centers

Application scenario:

- Large-scale AI model training data storage

- Model parameter storage for inference service

- Multimodal data lake

Core advantages:

- Unlimited Scalability: As data volume grows, simply add nodes

- High Throughput: RLNC parallel reading to meet GPU bandwidth requirements

- High reliability: Double error correction, training data will never be lost

Performance indicators:

- Sequential read: 100+ GB/s (1000 node cluster)

- Random IOPS: 10M+ (thanks to distributed architecture)

Edge computing

Application scenario:

- Autonomous driving data recording

- Industrial IoT data collection

- 5G edge cache

Core advantages:

- Elastic deployment: Dynamically increase or decrease nodes according to demand

- Local Computing: In-memory computing reduces data handling

- High reliability: Node failure does not affect the overall service

Typical configuration:

- Vehicle mounted: 16 nodes, 16TB effective capacity

- Factory: 64 nodes, 64TB effective capacity

Cold-data archiving

Application scenario:

- Enterprise data compliance archiving

- Long-term storage of medical images

- Permanent storage of scientific research data

Core advantages:

- Super long life: RLNC automatically repairs media aging

- Low cost: Using QLC/PLC NAND, high capacity density

- Zero Maintenance: The system heals itself without manual intervention

Economy:

- Storage cost: < $0.01/GB/year

- Data retention period: 100+ years (theoretical value)

Competitive advantages

Dimensions	Traditional SSD	RAID Array	Unlimited Capacity Storage
Scalability	Fixed capacity	Limited expansion	Unlimited expansion
Reliability	Single point of failure	Time-consuming to rebuild	Automatic self-healing
Performance	Limited by master	Limited by controller	Linear expansion
Cost	High (requires master control)	High (requires RAID card)	Low (decentralized)
Maintenance	Requires replacement	Requires rebuild	Zero maintenance

Core conclusion: Unlimited capacity storage is the next paradigm shift in storage technology

11. Prototype test cases

Test-case overview

In order to verify the feasibility of the unlimited capacity storage system, the following functional tests need to be completed on the FPGA prototype. The test is divided into four stages: basic functions, RLNC encoding and decoding, Mesh network, and system integration.

Phase 1: basic functionality

Test number	Test name	Test objectives	Passing criteria
T1.1	Flash read and write	Verify the correctness of the NAND interface	Write data = Read data
T1.2	SerDes loopback	Verify high-speed interface	Bit error rate < 10^-12
T1.3	GF(2^8) ALU	Verify finite field operations	All operation results are correct
T1.4	LFSR generation	Verify random number quality	Pass NIST test

Phase 2: RLNC encoding and decoding

Test number	Test name	Test goal	Passing criteria
T2.1	Single-generation coding	Verify the correctness of RLNC coding	k packets can restore the original data
T2.2	Recoding	Verify regenerated coding	New coding packets are linearly independent
T2.3	Gaussian elimination	Verify decoding correctness	Decoded data = original data
T2.4	MRD Verification	Validation Error Detection	100% Detection of Injection Errors

Phase 3: Mesh networking

Test number	Test name	Test goal	Passing criteria
T3.1	Node discovery	Verify handshake protocol	New node < 1s to join
T3.2	Route Forwarding	Verify Packet Forwarding	TTL Decremented Correctly
T3.3	Flow control mechanism	Verification Credit management	No packet loss, no deadlock
T3.4	Fault recovery	Verification node failure handling	Automatic data recoding

Phase 4: system integration

Test number	Test name	Test objectives	Passing criteria
T4.1	NVMe interface	Verify host compatibility	Identified as standard NVMe device

Test number	Test name	Test objectives	Passing criteria
T4.2	Sequential writes	Verify write performance	> 1GB/s (4 nodes)
T4.3	Sequential Read	Verified Read Performance	> 2GB/s (4 nodes)
T4.4	Random Read and Write	Validation IOPS	> 100K IOPS
T4.5	Hot expansion	Verify dynamic expansion	Node added online successfully
T4.6	Data persistence	Verify power outage protection	Data integrity after power outage

Milestone: After completing all tests, you can enter the mass production preparation stage