Excellent. You're not allowing gaps. That's exactly how real LRU engineering should be done.

No — Phase 3 is NOT fully complete yet.

What I gave you was architectural direction.
For a **multi-card embedded LRU**, Phase 3 must go deeper into:

- Deterministic architecture
- Data ownership
- Scheduling strategy
- Inter-card protocols
- Fault containment
- Memory strategy at system level
- Maintainability & field behavior

Let me complete Phase 3 properly.

🔥 PHASE 3 — LRU System Architecture & Reliability (Completed Properly)

Objective:
Transform you from "board-level firmware engineer" into "LRU system architect".

This is where engineering maturity is built.

WEEK 13 — Architectural Discipline

You must design system structure properly.

13.1 Strict Layer Separation

Correct layering:

```
Application
  ↓
Service / Middleware
  ↓
Driver Abstraction Layer
  ↓
Register-Level Drivers
```

```
    ↓
Hardware
```

Rules:

- Application cannot access registers
- Drivers cannot contain business logic
- No circular dependencies
- Each module must have a clear API

You must implement:

- Header file interfaces
- Internal static functions
- Module-private data

13.2 Data Ownership Model

In multi-card LRU:

Who owns data?

- Sensor card?
- Master card?
- Communication card?

You must define:

- Single source of truth
- Write authority
- Read-only mirrors

No uncontrolled global variables.

13.3 Scheduling Model

You must choose:

1. Super-loop (cooperative scheduler)
2. Time-triggered cyclic executive
3. Interrupt-driven model
4. RTOS (later phase)

For safety LRU, often:

Time-triggered cyclic scheduler.

You must design:

- Task execution slots
- Worst-case execution time (WCET)
- Jitter limits

WEEK 14 — System-Level State Machine (Deep)

Not LED-level state machine.

System-level state machine.

You must define:

States:

- POWER_OFF
- POWER_UP
- INITIALIZATION
- PBIT_RUNNING
- READY
- OPERATIONAL
- DEGRADED
- FAULT_LATCHED
- MAINTENANCE

For each state define:

- Entry actions
- Exit conditions
- Allowed transitions
- Fault reactions

No undefined transitions allowed.

WEEK 15 — Built-In Test (BIT) Engineering

This is critical in LRU.

15.1 Power-On BIT (PBIT)

Must test:

- RAM test (March algorithm basic version)
- Flash CRC
- Peripheral register readback
- Communication interface sanity

Failure must:

- Log error
- Prevent operational entry if critical

15.2 Continuous BIT (CBIT)

Must run during operation:

- Periodic memory checks
- Communication heartbeat
- ADC sanity
- Sensor plausibility checks

Must NOT:

- Disturb real-time performance
- Cause false alarms

15.3 Initiated BIT (IBIT)

Triggered by maintenance command.

Must:

- Isolate component under test
- Run deeper diagnostics
- Report structured result

WEEK 16 — Fault Management Strategy

This separates mature systems from hobby firmware.

16.1 Fault Classification

You must define:

- INFO
- WARNING
- MINOR FAULT
- MAJOR FAULT
- CRITICAL FAULT

Each must define:

- Log?
- Degrade?
- Shutdown?

16.2 Fault Containment

If Card 3 fails:

- Does entire LRU stop?
- Or isolate that card?

You must design:

- Watchdog per card
- Communication timeout
- Safe isolation mechanism

16.3 Fault Logging

You must design:

- Non-volatile error log
- Timestamp or counter
- Clearable by maintenance

Log format must be deterministic and structured.

WEEK 17 — Health Monitoring

You must monitor:

- Voltage rails
- Temperature
- Communication integrity
- Reset causes
- Watchdog resets

Implement:

Periodic health task.

Threshold strategy:

- Warning threshold
- Critical threshold
- Hysteresis to avoid oscillation

WEEK 18 — Multi-Card Coordination

Now we go full LRU-level thinking.

18.1 Heartbeat Protocol

Each card must:

- Send heartbeat at fixed interval
- Contain health summary
- Contain error flags

Master must:

- Detect timeout
- Trigger degrade state

18.2 Synchronization

If cards share data:

- Define update rate

- Define data validity window
- Define stale-data policy

18.3 Startup Coordination

Power sequencing:

- Master waits for slaves ready
- Slaves send ready signal
- Only then system transitions to operational

WEEK 19 — Configuration Management

LRU must support:

- Software versioning
- Configuration parameters
- Calibration constants

You must implement:

- Version structure
- Parameter storage in flash
- CRC validation of config

WEEK 20 — Safe-State Engineering

You must define:

If catastrophic fault happens:

What is safe state?

Examples:

- Shut down outputs
- Set relays to default
- Notify higher-level system

Never leave hardware in undefined state.

WEEK 21 — Maintainability & Field Behavior

Think long-term.

You must design:

- Bootloader update strategy
- Firmware update verification
- Rollback mechanism
- Debug hooks (non-intrusive)

Phase 3 Completion Criteria (True Completion)

You are Phase-4 ready only if:

- ✔ You can design layered architecture
- ✔ You design deterministic scheduler
- ✔ You design complete system state machine
- ✔ You design BIT (PBIT/CBIT/IBIT) properly
- ✔ You implement structured fault management
- ✔ You design multi-card heartbeat protocol
- ✔ You define safe-state strategy
- ✔ You manage configuration and versioning

If you can do all that — you are thinking like LRU architect.

✅ Phase 3 is NOW Fully Complete.

🚀 PHASE 4 — Verification, Validation & Certification Mindset

Now we move into the most misunderstood domain:

You will learn:

- Verification vs Validation
- Requirements traceability

- Test coverage strategy
- Structural coverage (statement, branch, MC/DC)
- Static analysis
- Code reviews
- Configuration control
- Documentation discipline
- Safety thinking (DO-178-like mindset without naming standard)

This is where engineers become system engineers.

If you are ready, I will now go into Phase 4 with the same depth and completeness.

Proceed?