Excellent.

Phase 4 is where most engineers break.

Up to Phase 3, you learned how to **build** an LRU.
Phase 4 teaches you how to **prove it is correct, safe, and reliable**.

This is the difference between:

- Hobby firmware
- Industrial embedded
- Aerospace-grade LRU

We now enter:

🔥 PHASE 4 — Verification, Validation & Certification Mindset

Duration: 8–12 Weeks
Objective: Make you capable of delivering certifiable, auditable, traceable firmware.

This is engineering discipline at its highest level.

WEEK 22 — Requirements Engineering (Foundation of Everything)

If requirements are wrong → system is wrong.

22.1 Types of Requirements

You must distinguish:

1. System Requirements
2. Hardware Requirements
3. Software Requirements
4. Derived Requirements
5. Interface Requirements

Each must be:

- Clear

- Testable
- Unambiguous
- Traceable
- Verifiable

Bad requirement:

> “System should respond quickly.”

Good requirement:

> “System shall respond to command within 50ms ±5ms.”

22.2 Requirement Attributes

Each requirement must have:

- Unique ID
- Description
- Source
- Verification method (Test / Analysis / Inspection / Review)
- Safety classification

22.3 Traceability Matrix

You must maintain:

```
Requirement → Design → Code → Test Case → Test Result
```

No orphan requirement allowed.

No unverified code allowed.

WEEK 23 — Software Design Documentation

Now you formalize architecture.

You must produce:

23.1 Software Architecture Document

Contains:

- Layer diagram
- Module breakdown
- Interfaces
- Scheduling model
- State machine diagrams
- Error handling strategy

23.2 Detailed Design Document (Per Module)

Each module must define:

- Purpose
- Inputs
- Outputs
- Assumptions
- Failure modes
- Interface functions
- Data structures

23.3 Interface Control Document (ICD)

For multi-card LRU:

- Packet formats
- Message timing
- Error codes
- Version compatibility rules

This prevents integration chaos.

WEEK 24 — Static Analysis & Coding Standards

You now eliminate defects before runtime.

24.1 Coding Standard Enforcement

You must follow strict rules:

- No dynamic memory (unless justified)
- No recursion
- No implicit type conversions
- All switches have default
- All pointers validated
- No magic numbers

24.2 Static Analysis Tools

You must understand what tools check:

- Null pointer dereference
- Dead code
- Memory leak
- Unreachable branches
- Arithmetic overflow
- MISRA violations

You must:

- Run static analysis
- Resolve warnings
- Document deviations properly

24.3 Defensive Programming Patterns

- Range checking
- Input validation
- Timeout handling
- Assertion strategy (only for development builds)

WEEK 25 — Unit Testing (Deep Level)

Every module must be testable in isolation.

25.1 Unit Test Philosophy

Test:

- Normal behavior
- Boundary conditions
- Error conditions
- Invalid inputs

25.2 Mocking Hardware

Since hardware is not always available:

- Replace register access with abstraction layer
- Use mock drivers
- Simulate failure injection

Example:

- Simulate ADC returning invalid voltage
- Simulate communication timeout

25.3 Code Coverage

You must measure:

- Statement coverage
- Branch coverage
- Condition coverage
- MC/DC (Modified Condition/Decision Coverage)

For critical LRU logic — high coverage is mandatory.

WEEK 26 — Integration Testing

Unit works alone.

Integration proves system works together.

26.1 Integration Strategy

Top-down

Bottom-up

Sandwich approach

You must define strategy before testing.

26.2 Hardware-Software Integration

Test:

- GPIO timing
- UART protocol correctness
- Interrupt priority conflicts
- Cross-card communication

26.3 Fault Injection Testing

You must deliberately:

- Corrupt packets
- Simulate power drop
- Force watchdog reset
- Simulate card failure

System must:

- Detect
- Log
- Transition state correctly

WEEK 27 — System Verification

Now you verify against system requirements.

27.1 Requirement-Based Testing

Each requirement must have:

- Test case ID
- Test procedure

- Expected result
- Actual result
- Pass/Fail

No test without requirement.

No requirement without test.

27.2 Timing Verification

You must measure:

- Interrupt latency
- Task execution time
- End-to-end response time
- Jitter

Use:

- Oscilloscope
- Logic analyzer
- Timestamp logs

27.3 Stress Testing

Test under:

- High temperature
- Voltage variation
- Continuous operation
- Maximum communication load

WEEK 28 — Validation

Verification = Did we build it right?

Validation = Did we build the right thing?

You must validate:

- Operational scenarios
- Edge cases

- Maintenance procedures
- Recovery after power loss
- Firmware update process

WEEK 29 — Configuration Management

You must manage:

- Version control
- Change control process
- Baseline freeze
- Build reproducibility

No undocumented change allowed.

29.1 Change Impact Analysis

Every change must answer:

- Which requirement affected?
- Which modules impacted?
- Which tests need rerun?

WEEK 30 — Safety & Hazard Thinking

You must learn to think like failure investigator.

30.1 Hazard Analysis

Identify:

- What could go wrong?
- What is worst-case consequence?
- How likely?
- How detectable?

30.2 Failure Mode Analysis

For each module:

- Failure mode
- Effect
- Detection mechanism
- Mitigation strategy

30.3 Safe State Verification

Test catastrophic scenario:

- Loss of communication
- Voltage out of range
- Critical memory corruption

System must enter defined safe state.

WEEK 31 — Documentation Discipline

Documentation must include:

- Software requirements spec
- Architecture design
- Detailed design
- Test plan
- Test report
- Traceability matrix
- Release note
- Known issues

If not documented — it does not exist.

WEEK 32 — Audit Readiness

If auditor asks:

- Show requirement traceability
- Show coverage report
- Show static analysis results

- Show fault handling strategy

You must respond confidently.

Phase 4 Completion Criteria

You are no longer just engineer.

You are certification-ready engineer if:

✓ You write testable requirements
✓ You maintain full traceability
✓ You design test cases for every requirement
✓ You measure code coverage
✓ You perform static analysis
✓ You document deviations
✓ You perform structured fault injection
✓ You maintain configuration control
✓ You think in safety terms

Transformation Summary

Phase 1 → Embedded C Engineer
Phase 2 → Board-Level Engineer
Phase 3 → LRU System Architect
Phase 4 → Certification-Ready Embedded Professional

Next Phase Options:

- Phase 5 → Advanced Embedded (RTOS, DMA, optimization, performance tuning)
- Phase 6 → Leadership & Technical Ownership in LRU Programs
- Or simulate a complete LRU project and I assign you role as firmware owner

Your move.