Good. You're thinking correctly — no gaps allowed.

No — Phase 2 is NOT fully complete yet.

What I gave you was structure + direction.
For an **LRU multi-card embedded system**, Phase 2 must go much deeper.

I will now **complete Phase 2 properly**, then move to Phase 3.

This will be engineering-grade depth.

🔥 PHASE 2 (Completed Properly)

Title: Hardware-Oriented Embedded C on Real LRU Hardware
Duration: ~6–8 Weeks

Objective:
Convert you from "C programmer" into "board-level embedded engineer".

This phase is about:

- Real hardware
- Real debugging
- Real faults
- Real timing problems
- Real system interaction

WEEK 5 — Toolchain, Boot & Startup (Deep)

Before GPIO, before UART — you must understand how your firmware even starts.

5.1 Boot Process Understanding

When LRU powers ON:

1. Power rails stabilize
2. Reset line released
3. MCU fetches reset vector
4. Stack pointer initialized
5. Startup code runs
6. .data copied to RAM
7. .bss zeroed
8. main() is called

You must understand:

- Startup assembly file

- Vector table structure
- Reset handler
- Weak ISR definitions

If you don’t understand this — debugging boot issues becomes impossible.

5.2 Linker Script (Critical)

You must learn:

- Flash origin
- RAM origin
- Section placement
- Stack size definition
- Heap size definition

You must:

- Modify stack size
- Observe memory overflow error
- Read .map file and analyze memory usage

5.3 Build System

You must know:

- What each compiler flag does
- Optimization effects on debugging
- Warnings and how to eliminate them
- Why `-Wall -Wextra` must be clean

WEEK 6 — GPIO (Done Like a Professional)

Not just blinking LED.

You must understand:

6.1 Electrical Layer

- Push-pull vs open-drain
- Pull-up / pull-down resistors
- Input floating behavior
- Output drive strength

Firmware must match hardware schematic.

6.2 Proper Driver Design

Not:

```
GPIOA->DATA = 0xFF;
```

But:

- Config structure
- Init function
- Error return
- Pin abstraction

Example:

```
typedef struct {
    uint8_t pin;
    uint8_t direction;
    uint8_t pull;
} gpio_config_t;
```

6.3 Validation

You must:

- Measure output using multimeter
- Verify switching speed using oscilloscope
- Compare waveform with expected timing

If waveform is wrong:

- Check clock config
- Check peripheral enable
- Check register settings

This is real embedded debugging.

WEEK 7 — Interrupts (Deep Dive)

Interrupts are where juniors break systems.

7.1 Interrupt Architecture

You must understand:

- NVIC (or equivalent)
- Interrupt priorities
- Preemption
- Nested interrupts

7.2 ISR Design Rules

ISR must:

- Be short
- Avoid printf
- Avoid blocking loops
- Only set flags or push data to buffer

7.3 Concurrency & Race Conditions

Classic problem:

Main loop modifies buffer
ISR modifies buffer

You must understand:

- Critical section
- Disable interrupt scope
- Atomic operations
- Volatile is not synchronization

7.4 Debugging Interrupt Problems

Symptoms:

- System hangs
- Random reset
- Data corruption

You must:

- Inspect stack overflow
- Inspect vector table
- Check wrong interrupt priority
- Check unhandled ISR

WEEK 8 — Timers & Real-Time Behavior

Embedded LRU requires deterministic timing.

8.1 Timer Configuration

- Clock source
- Prescaler
- Auto reload
- Interrupt generation

8.2 Timing Accuracy

You must calculate:

```
Timer frequency = MCU clock / prescaler
```

Measure with oscilloscope.
Compare real vs theoretical.

8.3 Watchdog Timer

LRU must not freeze.

You must:

- Configure watchdog
- Refresh properly
- Test failure case (intentionally hang system)

WEEK 9 — UART & Communication Robustness

Not just sending “Hello”.

9.1 Driver Modes

- Polling
- Interrupt
- DMA (if available)

Understand tradeoffs.

9.2 Buffer Management

Implement:

- Circular buffer
- Overflow detection
- Timeout handling

9.3 Error Handling

Detect:

- Parity error
- Framing error
- Overrun error

Simulate cable disconnect.

WEEK 10 — Multi-Card Communication (LRU Context)

Now we move into real LRU territory.

10.1 Backplane Communication

Understand:

- Bus topology
- Master-slave vs peer-to-peer
- Addressing scheme

10.2 Protocol Design

Design simple protocol:

Packet format:

```
[HEADER][SOURCE][DEST][LEN][DATA][CRC]
```

You must implement:

- Packet parser
- CRC calculation
- Timeout logic
- Retransmission

10.3 Fault Injection Testing

Simulate:

- Corrupted CRC
- Timeout
- Partial packet
- Noise

System must not crash.

WEEK 11 — Power & Reset Handling

LRU reliability depends heavily on this.

You must understand:

- Brownout reset
- Power sequencing
- Reset reason register
- Safe startup state

Test:

- Power cycling rapidly
- Sudden power drop
- Warm reset

WEEK 12 — Hardware Debugging Skills

You must be comfortable with:

- Oscilloscope
- Logic analyzer
- Multimeter
- Reading schematic

You must isolate:

Firmware issue vs hardware issue.

Example:

- No UART output.
  Is it:
- Wrong baud rate?
- Clock misconfigured?
- TX pin wrong?

- Transceiver disabled?
- Hardware fault?

You must investigate systematically.

PHASE 2 FINAL REQUIREMENTS

You are Phase-3 ready only if:

✔ You understand boot process
✔ You can modify linker script
✔ You can write driver from datasheet
✔ You debug interrupt race conditions
✔ You validate timing physically
✔ You design simple communication protocol
✔ You handle error cases safely
✔ You isolate firmware vs hardware fault

If any of these are weak → you are not ready for system-level LRU design.

✅ Phase 2 is NOW Complete.

🚀 PHASE 3 — LRU System Architecture & Reliability Engineering

Now we move from “board engineer” to “LRU system engineer”.

This is where you become valuable in aerospace/defense/industrial embedded.

PHASE 3 — SYSTEM-LEVEL THINKING

Duration: 6–8 Weeks

Objective:

You will learn:

- LRU architecture design
- Multi-card coordination
- Built-In Test (BIT)
- Fault tolerance
- State management
- Health monitoring

- Safe state design

WEEK 13 — LRU System Architecture

You must design:

```
+------------------+
| Application Layer|
+------------------+
| Service Layer    |
+------------------+
| Driver Layer     |
+------------------+
| Hardware         |
+------------------+
```

Rules:

- Application never touches registers
- Drivers never contain business logic

WEEK 14 — State Machine Design (LRU-Level)

Your LRU must have states:

- POWER_UP
- INIT
- SELF_TEST
- OPERATIONAL
- DEGRADED
- FAULT
- MAINTENANCE

You must design:

- Transition rules
- Fault triggers
- Recovery logic

State diagram mandatory.

WEEK 15 — Built-In Test (BIT)

Types:

1. Power-On BIT (PBIT)

2. Continuous BIT (CBIT)
3. Initiated BIT (IBIT)

You must implement:

- RAM test
- Flash CRC test
- Communication self-test
- Peripheral register check

BIT must not falsely fail.

WEEK 16 — Fault Handling Strategy

Design:

- Error classification
- Recoverable vs non-recoverable
- Logging mechanism
- Fault escalation

System must:

- Not enter undefined state
- Not hang silently

WEEK 17 — Health Monitoring

Monitor:

- Voltage levels
- Temperature sensors
- Communication link status
- Watchdog resets

Implement:

- Periodic health task
- Threshold detection
- Graceful degradation

WEEK 18 — Multi-Card Coordination

In multi-card LRU:

- Master card
- Slave cards
- Heartbeat mechanism
- Synchronization

You must design:

- Card alive monitoring
- Failover mechanism
- Safe isolation of faulty card

PHASE 3 Completion Criteria

You are no longer junior if:

✔ You can design LRU architecture
✔ You think in states
✔ You design BIT correctly
✔ You handle faults predictably
✔ You design communication robustly
✔ You think in safety terms

You now moved from:

Coder → Firmware Engineer → Board Engineer → LRU System Engineer

---

Now tell me:

Do you want me to continue into:

- 🔬 Phase 4 — Verification, Validation & Certification Mindset
- 🧠 Phase 5 — Advanced Embedded (RTOS, DMA, performance tuning)
- Or simulate a real LRU project and assign you responsibilities?