Short answer first — **yes, Phase-0 is fully covered**.

Long answer (important):
Phase-0 is **not about volume**, it's about **orientation + thinking discipline**.
Everything that *must* exist before engineering work begins **has been covered**:

✔ mindset reset
✔ system purpose
✔ failure thinking
✔ boundaries
✔ criticality
✔ senior evaluation checks
✔ outputs
✔ pass/fail consequences

There is **nothing else to add** without leaking into Phase-1.
So we **must move forward**, exactly like a real program would.

🔵 PHASE-1 — SYSTEM DECOMPOSITION & REQUIREMENT OWNERSHIP

(This is where engineers are separated from "document readers")

If Phase-0 answers **"Why does this exist?"**
👉 Phase-1 answers **"What exactly must this system GUARANTEE?"**

1️⃣ What Phase-1 REALLY Is

❌ Fresher misunderstanding

> "Phase-1 means start studying hardware/software."

✅ Reality

> **Phase-1 is about converting vague system intent into enforceable engineering responsibilities.**

In avionics programs, this is called:

- Requirement ownership
- System decomposition
- Functional allocation

If done badly → **design rework, late failures, certification pain**

2️⃣ Phase-1 OBJECTIVES (Non-Negotiable)

By the end of Phase-1, you must be able to:

1. Break the system into **clear functional blocks**
2. Identify **who owns which requirement**
3. Separate **system vs hardware vs software responsibilities**
4. Identify **derived requirements**
5. Detect **missing / ambiguous requirements**
6. Speak using **requirement language**, not guesses

If you cannot → you are **not ready to design**.

3️⃣ Phase-1 PROCEDURE (Step-by-Step, Real Industry Style)

🔹 Step 1: Functional Decomposition (Top-Down Only)

You take the **system existence statement** from Phase-0 and ask:

> "What functions must exist for this system to fulfill its purpose?"

Example: Mission Computer

High-Level Function
Data acquisition
Data processing
Data fusion
Data distribution
Health monitoring
Fault reporting
Mode management

⚠️ RULE:
No buses, no CPUs, no cards, no software yet.

If someone says "PowerPC" here → **Phase-1 fail**.

🔹 Step 2: Convert Functions into Functional Requirements

Now you map functions → **“shall” statements**.

Example:

- “The Mission Computer **shall acquire** data from avionics sensors”
- “The Mission Computer **shall process** data within defined latency”
- “The Mission Computer **shall provide** fault status to BIT system”

This is where you learn:

> **Specs describe guarantees, not implementations**

🔹 Step 3: Functional Flow Diagram (Critical Step)

You must draw a **functional flow**, not a block diagram.

```
Sensor Data
   ↓
Validation
   ↓
Processing
   ↓
Fusion
   ↓
Distribution
```

Why seniors care:

- Shows sequencing understanding
- Exposes timing assumptions
- Reveals hidden dependencies

Most freshers skip this → **instant red flag**.

🔹 Step 4: Allocate Responsibility (System vs HW vs SW)

Now comes a **career-defining skill**.

For each function, ask:

Function	System	HW	SW
Data acquisition	✔	✔	✔
Processing	✔		✔
Fault detection	✔	✔	✔

Function	System	HW	SW
BIT reporting	✓		✓

Key lesson:

> **System requirements never disappear — they flow downward**

🔹 Step 5: Identify Derived Requirements (Hidden Skill)

Specs are never complete.

Example:

- If latency is specified → clock accuracy is implied
- If availability is required → watchdog is implied
- If safety critical → redundancy is implied

These are **derived requirements**.

Seniors silently watch if you detect these.

🔹 Step 6: Requirement Classification (Professional Move)

You now classify each requirement as:

- Functional
- Performance
- Interface
- Safety
- Testability
- Maintenance

This directly impacts:

- Design decisions
- Verification planning
- Certification effort

🔹 Step 7: Detect Ambiguity & Raise Questions (Very Important)

You **must** produce a list like:

- “Latency not defined for degraded mode”
- “Reset behavior during mission not specified”
- “BIT response time unclear”

This is **not criticism**.
This is **engineering maturity**.

4️⃣ COMMON PHASE-1 FAILURES (Seen Daily)

❌ Failure 1: Jumping to Architecture

> “We can use VPX with Ethernet…”

→ You skipped responsibility definition.

❌ Failure 2: Treating Spec as Perfect

> “Spec says so, so it must be correct.”

→ Specs are *contracts*, not truths.

❌ Failure 3: Mixing Function with Implementation

> “Processing means PowerPC.”

→ No. Processing means **guarantees**.

❌ Failure 4: No Derived Requirements

> “Everything is already written.”

→ Impossible in real avionics.

5️⃣ PHASE-1 EVALUATION CHECKS (Used by Seniors)

They may ask:

🔍 Check 1

> “Which function is most timing-critical?”

🔍 Check 2

"Which requirement is hardest to verify?"

🔍 Check 3

"What breaks if this function is removed?"

🔍 Check 4

"Which requirements drive hardware choices?"

If you answer clearly → you pass.

6️⃣ PHASE-1 OUTPUTS (MANDATORY)

By end of Phase-1, you **must** have:

1. Functional decomposition list
2. Functional flow diagram
3. Requirement-to-function mapping
4. Responsibility allocation table
5. Derived requirements list
6. Open issues / ambiguities list

No outputs → no trust.

7️⃣ PHASE-1 OUTCOMES

✅ If You Pass

- You're invited to architecture discussions
- Your opinions are considered
- You move into **real engineering**

❌ If You Fail

- You get task-level work only
- Designs are "given" to you
- Growth stalls quietly

8️⃣ PHASE-1 GOLDEN RULES

1. **Functions before architecture**
2. **Requirements before components**
3. **Ambiguity must be exposed**

4. **Ownership defines accountability**
5. **If you can’t map it, you don’t own it**

9️⃣ PHASE-1 COMPLETION CHECK

You are **Phase-1 complete** if you can answer:

> “Which requirement keeps you awake at night — and why?”

Next logical step

👉 **Phase-2: Architecture Definition & Technology Trade-offs**

This is where:

- PPC vs ARM debates happen
- VPX / backplane choices emerge
- Redundancy philosophies are decided

If you want, I can:

- Continue directly to **Phase-2**
- Apply Phase-1 **specifically to Mission Computer**
- Create **Phase-1 evaluation tasks for freshers**
- Simulate a **real design review interrogation**

Just tell me how you want to proceed.