Phase 1: Die „Elefanten im Raum“ (Hardware-Kritik)

Prüfer werden sofort auf die fehlende Zertifizierung des ESP32 hinweisen. Bereiten Sie diese Antworten auswendig vor.

- **Kritik:** „Der ESP32 ist ein Bastler-Chip und erfüllt nicht ISO 26262.“
  - **Antwort (Die Pivot-Strategie):** „Das ist korrekt. Der ESP32 ist ein QM-Chip (Quality Managed). Ich nutze ihn als *Demonstrator für Prozesse*. Mein Fokus lag darauf, zu zeigen, wie man *Software* nach ASIL-Kriterien strukturiert (V-Modell, MISRA), unabhängig von der Hardware. Für die Serie würde der Code auf einen Infineon Aurix oder NXP S32 portiert.“
- **Kritik:** „Warum kein Lockstep?“
  - **Antwort:** „Der ESP32 hat zwei Kerne, nutzt aber SMP (Symmetric Multiprocessing), kein Lockstep.

    Im Lockstep führen zwei Kerne denselben Befehl zeitgleich aus und vergleichen das Ergebnis. Beim ESP32 arbeiten sie unabhängig. Für ASIL-D wäre Lockstep zwingend, hier zeige ich eine ASIL-B/QM-Architektur.“
- **Kritik:** „Was passiert bei einem Bit-Flip im RAM?“
  - **Antwort:** „Der ESP32 hat kein ECC-RAM (Error Correction Code). In einem echten ASIL-System müsste dies hardwareseitig korrigiert werden. In meiner Software fange ich Logikfehler ab, aber Hardwarefehler bleiben das Risiko dieses Prototyps.“

Phase 2: Methodik & C++ (Warum nicht C?)

Hier punkten Sie mit technischer Tiefe.

- **Frage:** „C++ im Auto? Das verbraucht zu viel Speicher und ist undeterministisch!“
  - **Antwort:** „Nicht modernes Embedded C++.
    1. **Speicher:** Ich nutze keine Exceptions (-fno-exceptions) und keine RTTI.
    2. **Heap:** Nach der Initialisierung (main) gibt es kein new oder malloc mehr. Alle Container (std::array) liegen auf dem Stack oder im BSS.
    3. **Determinismus:** Durch constexpr verlagere ich Berechnungen in die Compile-Zeit. Das *spart* Laufzeitzyklen gegenüber C.“
- **Frage:** „Warum std::optional statt Pointer?“
  - **Antwort:** „Ein Pointer kann nullptr sein, aber auch uninitialisiert oder ungültig. std::optional zwingt mich zur Prüfung (has_value()). Es macht die API ehrlich: Der Rückgabetyp sagt ‚Ich könnte scheitern‘. Das verhindert den klassischen Null-Pointer-Dereferenzierungs-Crash.“

Phase 3: Architektur & Tools

Beweisen Sie, dass Sie Systemdenken beherrschen.

- **Frage:** „Warum PlatformIO und nicht Makefiles?“

    - **Antwort:** „Wegen der Reproduzierbarkeit (Configuration Management nach ISO 26262). Die platformio.ini friert Compiler-Versionen und Linter-Regeln ein. Ein make hängt oft von der lokalen Umgebung des Entwicklers ab."
- **Frage:** „Erklären Sie Ihre Schichtenarchitektur."
    - **Antwort:** „Ich folge einer strikten 3-Schichten-Architektur:
        1. **MCAL (Microcontroller Abstraction Layer):** Kapselt esp-idf. Wenn ich auf Infineon wechsle, tausche ich nur diesen Ordner.
        2. **Service / RTE:** Abstrahiert Signale (z. B. ‚Spannung' statt ‚ADC-Wert').
        3. **Application:** Enthält reine Logik ohne Hardware-Abhängigkeit. Sie ist theoretisch auf dem PC testbar."

Phase 4: Die „Killer-Fragen" (Szenarien)

Wenn der Prüfer tief bohren will:

- **Szenario:** „Ihr Sensor fällt aus. Wie reagiert Ihr System?"
    - **Lösung:** „Der MCAL liefert ein leeres std::optional. Der Service-Layer erkennt das, setzt den Systemstatus auf ‚Degraded' und die Applikation fährt in den ‚Safe State' (z. B. Motor aus). Es gibt kein undefiniertes Verhalten."
- **Szenario:** „Wie garantieren Sie Echtzeit auf dem ESP32?"
    - **Lösung:** „FreeRTOS ist präemptiv, aber nicht hart-echtzeitfähig wie ein OSEK-OS. Ich nutze feste Prioritäten und verzichte auf blockierende Aufrufe in High-Prio-Tasks. Für harte Deadlines (< $1\,\mathrm{ms}$) wäre der ESP32 ungeeignet."

Phase 5: Live-Demo (Vorbereitung)

Falls Sie den Code zeigen müssen:

1. **Der „Aha"-Effekt:** Öffnen Sie main.cpp. Löschen Sie eine geschweifte Klammer oder ein Semikolon. Speichern Sie.
    - *Zeigen Sie:* PlatformIO markiert den Fehler sofort rot (Linter), noch *bevor* Sie kompilieren.
    - *Kommentar:* „Das ist meine erste Verteidigungslinie: Statische Analyse in Echtzeit."
2. **Der Clean-Code Beweis:** Navigieren Sie zu lib/Mcal_System.
    - *Zeigen Sie:* Nur hier gibt es #include "driver/gpio.h".
    - *Kommentar:* „Sehen Sie, die Applikation weiß nicht, dass sie auf einem ESP32 läuft. Das ist echte Portierbarkeit."