1. Systemdokumentation (Die Architektur)

Zielgruppe: Software-Architekten, Reviewer.

Zweck: Nachweis der statischen Struktur und der Design-Entscheidungen.

Struktur-Vorlage:

1.1 Architekturschaubild (Schichtenmodell)

Beschreiben Sie die strikte Trennung, die wir im lib/ Ordner angelegt haben.

- **Application Layer (App_Core):** Enthält die reine Logik (z. B. Zustandsautomaten). Sie ist *hardware-agnostisch*.
  - *Regel:* Darf keine esp_-Header inkludieren.
- **Service Layer / RTE (Srv_Monitor):** Die Middleware. Sie entkoppelt die Applikation von der Hardware.
  - *Funktion:* Hier findet das Zeitmanagement (10ms Zyklus) statt.
- **MCAL (Mcal_System):** Microcontroller Abstraction Layer.
  - *Funktion:* Nur hier wird auf Register oder ESP-IDF-Treiber zugegriffen.

1.2 Schnittstellen-Definition

Dokumentieren Sie, wie die Schichten kommunizieren.

- **Synchron:** Funktionsaufrufe von oben nach unten (App $\rightarrow$ Srv $\rightarrow$ Mcal).
- **Datenfluss:** Nutzung von std::optional oder Referenzen statt roher Pointer.
- **Beispiel:**„Die Funktion readVoltage() gibt ein std::optional<float> zurück. Ein leerer Rückgabewert signalisiert einen physikalischen Sensorfehler (z. B. Kabelbruch), der im Service-Layer behandelt wird.“

1.3 Design-Constraints (Safety-Regeln)

Listen Sie auf, welche Regeln durch die platformio.ini und .clang-format erzwungen werden:

- Keine dynamische Speicherallokation (Heap) nach der Initialisierung.
- Keine Exceptions (-fno-exceptions).
- Alle Warnungen sind Fehler (-Werror).

---

2. Benutzerhandbuch (Der Betrieb)

Zielgruppe: Entwickler, Integratoren.

Zweck: Reproduzierbarkeit der Build-Umgebung sicherstellen.

Struktur-Vorlage:

2.1 Voraussetzungen (Prerequisites)

- **Hardware:** Seeed Studio XIAO ESP32-S3.
- **IDE:** VS Code mit Extension „PlatformIO IDE".
- **Toolchain:** Wird automatisch durch PlatformIO verwaltet (Espressif 32 Platform v6.x, GCC v11+).

2.2 Installation & Build

Verwenden Sie imperative, klare Anweisungen.

1. Repository klonen: git clone ...
2. VS Code öffnen: code .
3. Abhängigkeiten installieren: Warten, bis PlatformIO die Indexierung abgeschlossen hat.
4. Build ausführen:
   Bash
   pio run

   Erwartetes Ergebnis: „SUCCESS" (Dauer ca. 1–2 Minuten).

2.3 Statische Analyse ausführen (Quality Gate)

Erklären Sie, wie man den Compliance-Check startet.

- Befehl: pio check
- *Interpretation:* Die Ausgabe darf keine Fehler der Kategorien High oder Medium enthalten. Falls Warnungen auftreten, müssen diese entweder behoben oder (mit Begründung) in der Konfiguration unterdrückt werden.

---

3. Abschlussbericht (Die kritische Distanz)

Zielgruppe: Prüfer, Senior Engineers, HR.

Zweck: Einordnung der Leistung. Hier zeigen Sie, dass Sie den Unterschied zwischen „Basteln“ und „Engineering“ verstehen.

Dies ist der wichtigste Teil. Nutzen Sie hier eine **Tabelle zur Abgrenzung**.

3.1 Methodik

Beschreiben Sie den Ansatz:

> „Es wurde ein 'Quality Managed' (QM) Demonstrator auf Basis eines ESP32-S3 entwickelt. Ziel war die Anwendung von ISO-26262-Methodiken (V-Modell, Coding Standards, Statische Analyse) auf einer COTS-Hardware-Plattform.“

3.3 Fazit & Ausblick

- **Ergebnis:** Das Projekt beweist, dass moderne C++ Features (C++17) und strikte Linting-Regeln (MISRA-Anlehnung) die Code-Qualität signifikant erhöhen, auch ohne zertifizierte Hardware.
- **Limitation:** Für einen Serieneinsatz müsste die Hardware (Wechsel auf Aurix/S32K) und das OS (Wechsel auf AUTOSAR) getauscht werden. Die *Software-Architektur* (Schichtenmodell) bliebe jedoch erhalten.

3.2 Kritische Distanz: Prototyp vs. Serie

Hier beweisen Sie Fachkompetenz. Analysieren Sie die Lücken (Gaps).

Kriterium	Realisierung im Projekt (ESP32)	Anforderung ISO 26262 (ASIL-D Serie)	Bewertung der Lücke
Hardware	ESP32-S3 (Single/Dual Core, kein Lockstep)	Infineon Aurix / NXP S32 (Lockstep Cores, ECC-RAM)	**Kritisch:** Hardware erkennt Rechenfehler nicht selbstständig. Nur für QM/ASIL-A geeignet.
OS / Scheduler	FreeRTOS (Präemptiv, generisch)	AUTOSAR OS / SAFERTOS (Zeitlich/Räumlich partitioniert)	**Mittel:** FreeRTOS bietet keine garantierte Trennung (Memory Protection) zwischen Tasks.
Fehlerbehandlung	Software-Checks (if, std::optional)	E-Gas Monitoring (3-Ebenen-Konzept), Watchdogs	**Hoch:** Im Projekt fehlt eine unabhängige Überwachungsinstanz (externer Watchdog/PMIC).
Compiler	GCC (Open Source)	Zertifizierter Compiler (z.B. Tasking, Green Hills)	**Formal:** GCC fehlt das „Tool Qualification Kit“ für den Sicherheitsnachweis.