


SAYKAL ELECTRONICS - EMBEDDED SYSTEMS INTERVIEW TEST - 02

	EMBEDDED SYSTEMS INTERVIEW TEST-2	Rev : v1.0/02.05.2025 Doc.Typ: Internal / External
---	--------------------------------------	---

DOCUMENT APPROVAL PAGE

Name	Title	Sign	Date
Prepared by			
@mirac aydogan	Embedded Software Team Lead		May 2, 2025
Reviewed by			
@Gökhan Tekinok	Embedded Software Engineer		May 2, 2025
Approved by			
Anil SIRMA	R&D Manager		May 2, 2025

TABLE OF CONTENTS

- 1. INTRODUCTION
 - Objectives of this Case Study:
 - Case Sections:
- 2. PROBLEM STATEMENT
 - 2.1. Overview
 - 2.2. Electrical Parameters
 - 2.3. Functional Requirements
 - Voltage & Current Control
 - Monitoring and Telemetry (via CAN)
 - Error Handling
- 3. REQUIREMENTS AND CONSTRAINTS
- 4. TARGET HARDWARE CONTEXT
 - 4.1. Assumed Hardware Environment
 - 4.2. Implementation Note
- 5. SYSTEM ARCHITECTURE - UML DIAGRAM
- 6. DBC FILE DESIGN

1. INTRODUCTION [↗](#)

This document presents a technical case study designed for evaluating embedded software engineering candidates in the field of power electronics. The aim is to assess the candidate's skills in embedded control systems, real-time programming, and power converter control algorithms.

This case study simulates a real-world control scenario that allows candidates to showcase their embedded software design capabilities, problem-solving skills, and knowledge of microcontroller peripherals and control theory.

Objectives of this Case Study: [↗](#)

- **Control Systems Understanding:** Evaluate the candidate's grasp of real-time closed-loop control for power converters.
- **Embedded Software Implementation:** Assess practical embedded coding abilities for timers, ADCs, interrupts, and PWM generation.
- **Signal Processing and Stability:** Test the understanding of system dynamics, PI control design, and timing constraints.
- **Code Quality and Structure:** Ensure professional coding standards and modular software architecture.

Case Sections: [↗](#)

- Problem Statement: Control objective and scenario overview
- Constraints and Requirements: System-level and coding limitations
- Evaluation Criteria: Defined performance and assessment metrics

This document supports a fair and transparent evaluation for selecting capable embedded software engineers.

2. PROBLEM STATEMENT [↗](#)

2.1. Overview [↗](#)

Design and implement embedded software for a closed-loop control system managing a non-isolated Buck Converter using a microcontroller. The software must regulate the output voltage to 24V, regardless of input voltage fluctuations or load changes, and must monitor and limit output current to prevent overloading.

After stabilizing the output voltage, the embedded software must also continuously measure and transmit key operational data over CAN Bus, following a user-defined message protocol (.dbc file).

2.2. Electrical Parameters [↗](#)

- **Input Voltage Range:** 36V – 60V
- **Target Output Voltage:** 24V
- **Maximum Output Current:** 10A
- **Nominal Output Power:** Up to 240W
- **PWM Frequency:** 20 kHz
- **Control Loop Frequency:** 10 kHz
- **ADC Resolution:** 12-bit
- **Control Method:** PI/PID-based duty cycle control
- **PWM Duty Cycle Range:** 0% – 95%

2.3. Functional Requirements [↗](#)

Voltage & Current Control [↗](#)

- Measure output voltage and output current using ADC channels

- Implement a PI or PID controller to compute new PWM duty cycle
- Adjust duty cycle based on control output
- Limit output current:
 - If output current > 10A, reduce duty cycle to maintain safe current
 - Log the overcurrent condition

Monitoring and Telemetry (via CAN) [↗](#)

- The system must monitor and broadcast the following key data periodically (e.g., every 100ms):
 - Input Voltage
 - Input Current
 - Output Voltage
 - Output Current
 - System Temperature
 - System State (e.g., IDLE, RUNNING, ERROR)
- Use CAN Bus message or messages to transmit the data
- Candidate is expected to:
 - Design meaningful signal names, scaling, and byte allocations
 - Include signal units and physical limits
 - The sizes (bit lengths) of the data fields may be determined by the candidate based on signal resolution and efficiency considerations.

Error Handling [↗](#)

- All abnormal conditions (overcurrent, ADC failure, PI instability) must be reported to a global error handler module
 - Errors must also be transmitted periodically via CAN as part of the system status
-

3. REQUIREMENTS AND CONSTRAINTS [↗](#)

All code must be written in C or C++ and must adhere to the internal coding standard **STD.COD.01 CODING STANDARD**, which includes:

- **Naming Conventions:** Descriptive names with consistent prefixes/suffixes
- **Code Style:** Consistent indentation, line spacing, and modular separation
- **Commenting:** Clarify control algorithms, register setups, and condition checks
- **Version Control:** Use Git with meaningful commit messages

4. TARGET HARDWARE CONTEXT [↗](#)

4.1. Assumed Hardware Environment [↗](#)

- **Microcontroller:** STM32, TI C2000 series or desired MCU
- **Peripherals:**
 - 2x ADC Channels (Vout, Iout)
 - PWM Timer
 - CAN Controller
 - Temperature sensor (internal or external)

4.2. Implementation Note [↗](#)

Candidates are not expected to simulate the hardware. However, functions, placeholders, or comments may be used to demonstrate how the system would interact with ADC inputs, CAN messaging, and system state handling in a real embedded environment.

5. SYSTEM ARCHITECTURE - UML DIAGRAM [↗](#)

As part of the design documentation, the candidate is expected to provide a simple UML Diagram that reflects the overall software architecture of the system. This diagram should help reviewers understand the module-level separation, control algorithm structure (e.g., PI/PID controller), and interactions within the embedded system.

6. DBC FILE DESIGN [↗](#)

Candidates are expected to deliver a .dbc a .dbc file containing at least one message based on CAN message software design for describing system CAN database. The .dbc file must include:

- **Message Names:** e.g., `DEVICE_STATUS1`, `DEVICE_STATUS2`
- **CAN ID:** e.g., 0x555 (Extended), 0x556 (Extended) ..
- **DLC:** 8 or more bytes
- **Signals:**
 - `Vin` – Input Voltage (e.g., 0–60V)
 - `Iin` – Input Current
 - `Vout` – Output Voltage
 - `Iout` – Output Current
 - `Temp` – System Temperature
 - `Status` – System State Enum (IDLE, RUNNING, FAULT, etc.)

Each signal must have:

- Physical units
- Scaling (factor, offset)
- Min/Max ranges
- Byte offset and bit position

You may use tools like [Vector DBC Editor], [Kayak], [CANDb++ (free)], or open-source alternatives.