Challenge 1: Embedded Programming, FreeRTOS, STM32, and ASIC

Problem Description

A motion controller containing a STM32 (SOC), a PCL6046 (ASIC) and using FreeRTOS, controlling a system similar to the one in this video.

Develop a feature: monitor four motors, halting any two before they collide. Using the ASIC's software limits function (Nippon Pulse PCL6046 manual, 6.13.2) as your starting point of this new feature.

- Define how this function would work.
- Use a dedicated thread; activate only when the feature enables.
- Define the inputs that are required from the user.
- Registers are read and written by the main thread using ReadReg(RegName) and WriteReg(RegName).
- The STM(32) and PCL6045 must communicate in a thread safe manner.

Goals:

- **Function Planning:** Develop a feature plan; you will use this plan for the rest of the prompt.
- **ASIC Register Configuration:** Research ASIC documentation, then read, modify, and write registers accordingly.
- **FreeRTOS Task Implementation:** Create at least two tasks using FreeRTOS. One task handles USB and SOC/ASIC communication; another handles periodic register access.
- **Inter-Task Communication:** Use a thread-safe communication mechanism for data transfer between tasks.
- **Synchronization and Safety:** If tasks share resources, use mutexes or semaphores for safe synchronization.
- **Code Clarity and Structure:** Ensure C/C++ code is clean, well-organized, and easy to understand. Use clear variable names and comments to document the function of each task and the main parts of your code's logic.
- **Technical Documentation:** Provide a brief README or code comments that explain your design decisions. Describe your methods for periodic timing, inter-task communication, and firmware logic simulation/testing. The documentation should be clear and user-friendly, as if explaining to another developer or a tester how the firmware operates.

Deliverables

To be Hosted to a Git of your choice:

- **Source Code:** C or C++ source code files implementing the feature, FreeRTOS tasks and any supporting code.
- **Function Planning:** provide a document or flowchart of how you will implement this feature.
- **Documentation of Approach:** A README or in-code documentation outlining the design. Explain how tasks communicate, any assumptions, and how you ensure thread safety. This documentation should be written in a clear, user-friendly manner, as if you are guiding a new developer through your code.

Time Estimate

Approximately 1 hour is expected for this challenge, assuming familiarity with FreeRTOS task creation and basic synchronization primitives. Focus on implementing a working solution rather than perfection due to the time constraint. Failing to complete the task within the hour incurs no penalty. Please provide what you have completed, including the function planning document.

Bonus (Optional Extra Credit)

Configurable Parameters: Make the system more flexible by allowing certain parameters to be easily changed. For example, the sampling interval or threshold could be defined in a configuration section (or read from a simulated config file). Demonstrating the ability to modify behavior without changing code (beyond initial setup) is a plus.

Additional Task or Feature: Introduce a third FreeRTOS task or an additional feature for realism. For instance, add a status LED blinker task that toggles an LED at a fixed rate to indicate the system is alive (heart-beat). Ensure this does not conflict with other task – this tests handling of multiple tasks using the same peripheral.