## 1. Core Embedded C / C++ Programming

**Q:** Difference between volatile and const volatile. Why is volatile needed?
**A:** `volatile`: tells the compiler the variable may change outside program control. `const volatile`: value changes unexpectedly but user code can't modify it. Used for read-only hardware registers.

**Q:** What happens if you don’t declare an ISR variable as volatile?
**A:** Compiler may optimize out repeated reads, leading to stale values.

**Q:** Difference between stack and heap memory in embedded systems.
**A:** Stack: local vars, fast, fixed size. Heap: dynamic allocation, risk of fragmentation.

**Q:** How are function pointers used in embedded systems?
**A:** For callbacks, ISRs, jump tables, modular driver APIs.

**Q:** Risks of using malloc/free in embedded systems?
**A:** Memory fragmentation, out-of-memory, non-deterministic execution.

**Q:** How to avoid race conditions with ISR/thread shared vars?
**A:** Use volatile, mutex/semaphore, or disable interrupts briefly.

## 2. Microcontrollers and Hardware Interaction

**Q:** How GPIOs work and how to configure input/output?
**A:** Registers control direction, pull-ups, drive strength.

**Q:** Difference between UART, SPI, and I2C.
**A:** UART: async, simple, 2 wires. SPI: fast, master/slave, 4 wires. I2C: 2 wires, addressable.

**Q:** How does DMA improve performance?
**A:** Transfers data without CPU, frees cycles, reduces power.

**Q:** Interrupt vs polling.
**A:** Interrupt: efficient, CPU reacts to events. Polling: simple, wastes cycles.

**Q:** What is a watchdog timer?
**A:** Hardware timer resets system if software hangs.

**Q:** How to reduce MCU power consumption?
**A:** Use sleep modes, lower clock, power down peripherals, interrupts instead of polling.

## 3. RTOS and Scheduling

**Q:** Bare-metal vs RTOS.
**A:** Bare-metal: loop + interrupts, simple. RTOS: tasks, scheduling, scalability.

**Q:** Explain priority inversion.
**A:** Low-priority holds resource needed by high-priority; solved by priority inheritance.

**Q:** Scheduling algorithms in RTOS.
**A:** Fixed-priority, round-robin, rate-monotonic, earliest deadline first.

**Q:** Difference between semaphores and mutexes.
**A:** Semaphore: counting or sync. Mutex: binary, ownership.

**Q:** Explain starvation vs deadlock.
**A:** Starvation: low-priority task never runs. Deadlock: circular resource wait.

## 4. Embedded Software Design

**Q:** How to design I2C driver?
**A:** Configure I2C peripheral, implement init/read/write, handle ACK, provide API.

**Q:** Steps for SPI driver with DMA.
**A:** Configure SPI + DMA, link buffers, trigger transfer, handle DMA complete interrupt.

**Q:** What is memory-mapped I/O?
**A:** Registers mapped to memory addresses, accessed like variables.

**Q:** Typical bootloader responsibilities.
**A:** Init system, verify firmware, load app, provide updates.

**Q:** How to debug random hangs?
**A:** Watchdog logs, UART traces, check leaks, use debugger/analyzer.

## 5. Testing and Debugging

**Q:** Debug with only UART/LEDs?
**A:** Blink LEDs, log over UART, toggle GPIOs for timing.

**Q:** Tools for embedded debugging?
**A:** JTAG, SWD, logic analyzer, oscilloscope, GDB.

**Q:** What is boundary scan (JTAG)?
**A:** IEEE 1149.1, tests interconnects via scan chains.

**Q:** Unit testing with hardware dependencies.
**A:** Use HAL, mocks, simulation.

## 6. Performance, Safety, Reliability

**Q:** Handling timing-critical code.
**A:** Use hardware timers, minimal ISRs, avoid unpredictable constructs.

**Q:** Hard vs soft real-time.
**A:** Hard: deadlines absolute (e.g., airbag). Soft: occasional misses allowed.

**Q:** Ensuring safety in automotive/medical systems.
**A:** ISO 26262, MISRA-C, redundancy, fail-safes, static analysis.

**Q:** What is debouncing?
**A:** Filter switch noise. HW: RC filter. SW: delays/state machine.

**Q:** Causes of memory corruption.
**A:** Buffer overflow, dangling pointers, race conditions, stack overflow.

## 7. System-Level and Architecture

**Q:** What is cache coherence?
**A:** Ensures caches stay consistent with RAM; critical with multicore/DMA.

**Q:** Designing for limited flash/RAM.
**A:** Optimize code, fixed buffers, constants in flash, avoid malloc.

**Q:** Interrupts and RTOS scheduling.
**A:** ISRs run first, then RTOS decides task. ISRs should signal tasks, not block.

**Q:** Trade-offs: 8-bit vs 16-bit vs 32-bit MCUs.
**A:** 8-bit: cheap, low power. 16-bit: mid. 32-bit: faster, more memory, more costly.

**Q:** Common communication protocols in automotive/IoT.
**A:** Automotive: CAN, LIN, FlexRay. IoT: UART, SPI, I2C, BLE, Wi-Fi, Zigbee.

## Glossary of Acronyms

**ISR:** Interrupt Service Routine – function that runs when an interrupt occurs.

**GPIO:** General-Purpose Input/Output – configurable pins for input/output.

**DMA:** Direct Memory Access – transfers data between peripherals and memory without CPU.

**MMIO:** Memory-Mapped I/O – hardware registers mapped into memory space.

**RTOS:** Real-Time Operating System – provides scheduling and task management.

**HAL:** Hardware Abstraction Layer – abstracts hardware details for portability.

**JTAG:** Joint Test Action Group – debugging and test interface (IEEE 1149.1).

**SWD:** Serial Wire Debug – ARM-specific 2-pin debug interface.

**CAN:** Controller Area Network – robust automotive bus.

**LIN:** Local Interconnect Network – low-cost automotive bus.

**BLE:** Bluetooth Low Energy – low-power wireless protocol.

**I2C:** Inter-Integrated Circuit – 2-wire serial communication protocol.

**SPI:** Serial Peripheral Interface – 4-wire synchronous protocol.

**UART:** Universal Asynchronous Receiver/Transmitter – serial TX/RX protocol.

**XTOS:** Xtensa Operating System – a lightweight kernel for Tensilica Xtensa CPUs, used under FreeRTOS in ESP32.