Embedded System Interview Question and Answer. Set – 5

Linkedin

Owner	UttamBasu
Author	Uttam Basu
Linkedin	www.linkedin.com/in/uttam-basu/

Topic: Bare-metal

1) **How would you implement a precise 10µs periodic task on a Cortex-M0 without using an RTOS?**

Answer:
Use a high-resolution timer (e.g., TIMx) running at 1 MHz. Set up output compare to trigger every 10 µs (ARR = 10). Use the ISR only to set a flag or trigger a DMA or PPI channel (if available). Keep ISR latency under 1 µs. Avoid SysTick for sub-millisecond timing.

2) **How do you guarantee deterministic interrupt latency on Cortex-M?**

Answer:
- Disable lower-priority interrupts using BASEPRI.
- Avoid tail-chaining delays by limiting nested interrupts.
- Use fast peripheral clock and shortest possible ISR code.
- Place critical ISRs in .fastcode section (e.g., TCM or ITCM).

3) **How would you detect and recover from a stuck ISR (e.g., due to unacknowledged interrupt)?**

Answer:
- Use a watchdog timeout counter incremented in the main loop.
- Reset watchdog only after all high-priority ISRs are confirmed inactive.
- Implement interrupt alive-check counters.
- As a fallback, force a system reset if ISR lockup is detected.

4) **How do you implement static stack overflow detection on ARM Cortex-M?**

Answer:
- Place a magic value (`0xDEADBEEF`) at the base of the stack in .ld file.
- Periodically check it (in idle loop or timer ISR).
- If overwritten, trigger a fault handler or system reset.

5) **How would you detect and handle use-after-write EEPROM corruption?**

Answer:
- Use double-buffered wear-leveled EEPROM entries.
- Include CRC and sequence numbers.
- On read, verify CRC; if corrupted, revert to previous version.
- Never overwrite in place — use erase-write-validate strategy.

6) **How do you protect critical configuration in Flash from unintentional overwrite?**

Answer:

- Lock Flash write regions via `FLASH->CR` or hardware option bytes.
- Use separate Flash sectors for configuration.
- Protect with CRC and shadow copy validation on boot.

7) How would you write a portable SPI driver for both STM32 and NXP Kinetis MCUs?

Answer:

- Use a hardware abstraction layer (`spi_hal_t`) with init, send, receive, and IRQ functions.
- Use conditional compilation or a HAL backend per platform.
- Keep ISR logic minimal and call `spi_hal_irq_handler()` from the MCU-specific vector.

8) How do you manage timing for a software I2C bit-banging driver at 400 kHz?

Answer:

- Use a 1 µs resolution hardware timer to toggle SCL/SDA precisely.
- Drive GPIO open-drain with correct setup/hold timing.
- Disable interrupts during transaction or use critical sections.

9) **What happens during ARM Cortex-M startup before main()?**

Answer:

- Stack pointer initialized from vector table.
- Reset handler runs:
  - `.data` copied from flash to RAM
  - `.bss` zero-initialized
  - Optional constructors run (C++)
- `main()` is then called.

10) **How do you implement a dual-bank firmware update without external flash?**

Answer:

- Allocate two firmware slots in internal flash.
- Use a bootloader to validate and switch banks.
- Store a version marker or CRC for validation.
- Use `SCB->VTOR` to switch vector table to new firmware region.

11) **How would you minimize power in a sensor node that samples every 5 minutes?**

Answer:

- Enter STOP/STANDBY mode.
- Use RTC or LPTIM to wake up.
- On wake-up: enable clock, sample sensor via DMA, go back to sleep.
- Use `__WFI()` between events.

12) **How do you implement a low-power debounce for a mechanical switch?**

Answer:

- Use `EXTI` interrupt on GPIO edge.
- Disable interrupt on first trigger.
- Start a timer for ~10 ms debounce window.
- On timer expiry, re-check pin state and re-enable `EXTI`.

13) **How do you capture context during a HardFault on Cortex-M?**

Answer:

- Trap HardFault_Handler.
- Extract stacked registers (PC, LR, PSR, R0–R3, R12) from the stack frame.
- Store to reserved RAM or EEPROM.
- Use `SCB->CFSR`, `HFSR`, `MMFAR`, and `BFAR` for deeper diagnosis.

14) **What if the MCU hangs randomly, but watchdog never fires?**

Answer:

- Check that the watchdog is properly configured and not being reset in ISR.
- Confirm the main loop is alive (heartbeat counter).
- Monitor for interrupt storms, stack overflow, or clock failures.

15) **What linker script modifications are needed to place a section at a specific Flash address?**

Answer:

```
.mysection 0x08020000 :
{
  KEEP(*(.my_section))
} > FLASH
```

Then in code:

```
__attribute__((section(".my_section"))) const uint8_t config_data[] = { ... };
```

16) **What are the dangers of enabling -O3 optimization in embedded code?**

Answer:

- Timing-sensitive code may break.
- Volatile accesses might be reordered/removed.
- Stack usage increases.
- Unused code may be stripped, breaking startup behavior.

17) **How do you implement atomic access to a shared variable in a Cortex-M0 system?**

Answer:

- Use `__disable_irq()` and `__enable_irq()` around critical section.
- Or use `LDREX/STREX` if supported (M3/M4 only).
- Keep critical sections short to avoid latency issues.

18) **How would you queue UART log messages without dynamic memory?**

Answer:

- Use a fixed-size ring buffer:
  - Producer (e.g., main or ISR) adds messages.
  - Consumer (UART TX complete ISR or idle loop) transmits.
- Buffer must handle wraparound and full/empty detection.

19) **How do you implement redundancy in sensor reading to detect faults?**

Answer:

- Sample multiple identical sensors.
- Cross-check using voting (2-out-of-3 logic).
- In case of mismatch, flag diagnostic error and switch to degraded mode.

20) What strategies would you use to make your firmware MISRA-C compliant?

Answer:

- Use static analysis tools (e.g., PC-lint, Coverity, Cppcheck).
- Avoid pointer arithmetic, recursion, uninitialized variables.
- Write HAL in safe wrappers.
- Enforce strict typedef usage and naming conventions.

21) How do you implement a secure firmware update mechanism over UART in a bootloader?

Answer:

- Authenticate update with SHA-256 or ECDSA signature.
- Encrypt firmware with AES (e.g., CTR mode).
- Verify CRC, signature, and version before flashing.
- Use double-buffered update with rollback protection.
- Prevent overwriting bootloader region via MPU or flash write protection.

22) How do you switch execution to a new firmware image stored in a different flash region?

Answer:

- Disable all interrupts.
- Set MSP from the image base:

```
__set_MSP(*(uint32_t*)NEW_FW_ADDR);
```

- Jump to reset vector:

```
((void (*)(void))(*(uint32_t*)(NEW_FW_ADDR + 4)))();
```

- Optionally, set `SCB->VTOR = NEW_FW_ADDR` before jump (on M3/M4/M7).

23) How would you validate an application image from a bootloader?

Answer:

- Store image CRC or hash at end of image.
- Compute CRC/hash on boot and compare.
- Include build version, timestamp, and magic header.
- Abort boot if image is invalid.

24) How do you use DMA to offload ADC sampling without CPU intervention?

Answer:

- Configure ADC in continuous or triggered mode.
- Set DMA source to `ADC->DR`, destination to RAM buffer.
- Use circular DMA mode for continuous sampling.
- Optional: Use DMA half/full complete interrupts for processing.

25) How would you double-buffer UART reception using DMA?

Answer:

- Allocate two buffers.
- Use DMA in circular mode with buffer size N.
- On half/full DMA transfer complete interrupt:
  - Process buffer.
  - Swap or reset buffer pointer.

26) What issues can arise from DMA writing to memory used by CPU concurrently?

Answer:

- Race conditions or stale data reads.
- Cache coherency issues (especially on Cortex-M7).
- Overwriting of unprocessed data.

  Mitigation:
- Use memory barriers `(__DSB())` , DMA flags, and buffer locks.
- Disable cache or use DTCM for shared buffers.

27) How do you safely write to a memory-mapped peripheral register?

Answer:

- Use volatile pointers:

```
*((volatile uint32_t*)REG_ADDR) = value;
```

- Avoid read-modify-write unless atomic.
- Use bit-banding (if supported) for single-bit access (e.g., Cortex-M3/M4).

28) What problems can occur with peripheral register access if volatile is omitted?

Answer:

- Compiler optimizes out read/writes.
- Multiple writes collapsed into one.
- Peripheral states ignored, leading to undefined behavior.

29) How do you implement a fault-resilient main loop with isolation for modules?

Answer:

- Each module has a watchdog/heartbeat.
- Use function pointers with sanity checks.
- Wrap risky modules in `setjmp()/longjmp()` or fault domains.
- Isolate memory with MPU (on M4/M7).

30) How do you detect a fault caused by an unaligned memory access on ARM Cortex-M?

Answer:

- Enable `UNALIGN_TRP` in `SCB->CCR`.
- Unaligned access triggers UsageFault.
- Handle in UsageFault_Handler and log offending address (`SCB->MMFAR`).

31) How do you trace code execution in real time without halting the CPU?

Answer:

- Use ITM/SWO with a debugger that supports it (e.g., STM32 ST-LINK, J-Link).
- Configure `ITM->TER` and `DWT->CTRL`.
- Use `ITM_SendChar()` or memory-mapped writes for logging.

32) What's the difference between BKPT and WFI, and when would you use each?

Answer:

- BKPT: triggers a breakpoint trap, halts CPU when debugger is attached.
- WFI: Wait For Interrupt — puts CPU into low power until IRQ.
  Use BKPT in debug builds for breakpoints.
  Use WFI in release for low-power idle loops.

33) How can the MPU be used to catch NULL-pointer dereference?

Answer:

- Set region `0x00000000`–`0x00000FFF` as no-access.
- Enable MPU, configure background region as R/W.
- NULL access triggers memory fault (HardFault or MemManage fault).

34) How do you enforce read-only access to a memory region at runtime?

Answer:

- Use MPU region with AP = read-only.
- Reconfigure MPU when access mode changes (e.g., bootloader vs app).
- Enforce privilege separation if using Thread/Handler modes.

35) How do you calibrate an internal RC oscillator (e.g., HSI or LSI)?

Answer:

- Use external clock reference (e.g., LSE or crystal).
- Measure ticks of internal clock over known period.
- Adjust trimming register (e.g., `RCC->ICSCR`) to correct frequency.

36) How do you generate a PWM signal with 0–100% duty cycle in hardware?

Answer:

- Use timer in PWM mode:
  - ARR = fixed period
  - CCRx = duty
- 0%: CCR = 0
- 100%: CCR = ARR + 1
- Configure output polarity to match requirements.

37) What are the side effects of changing PLL settings at runtime?

Answer:

- Glitches or instability if not done correctly.
- Must disable PLL before reconfiguration.
- Switch system clock to HSI temporarily during change.
- Ensure flash wait states and peripheral clocks are adjusted.

38) How do you implement a watchdog-based safety mechanism with a test window?

Answer:

- Use IWDG with known reload window.
- Refresh only after completing key safety checks.
- Missed refresh → system reset → safe state recovery.

39) How do you handle single-event upsets (bit flips) in critical RAM structures?

Answer:

- Use parity or ECC-protected RAM (if supported).
- Duplicate critical variables and cross-check.
- Periodic memory scrub and validation routines.

40) How would you structure firmware to comply with ISO 26262 (ASIL-B or higher)?

Answer:

- Decompose into safety & non-safety domains.
- Use MISRA-C, static analysis, and unit tests.
- Traceability from requirements to implementation.
- Implement diagnostic coverage (e.g., MCU self-test, CRC checks).
- Separate safety logic in isolated modules with MPU protection.

41) Can you run an RTOS on a Cortex-M0? What are the trade-offs?

Answer:
Yes, but with caveats:

- No BASEPRI, so critical sections disable all interrupts.
- Limited stack space—context switching must be minimal.
- Use lightweight RTOS (e.g., FreeRTOS with minimal config).
  Trade-off: Less interrupt flexibility, but can still achieve determinism.

42) How would you migrate a time-critical bare-metal ISR into an RTOS task?

Answer:

- Move only non-critical processing into task.
- Keep ISR minimal: set flag or notify task (e.g., via xTaskNotifyFromISR()).
- Use priority inheritance or task preemption if needed.
- Ensure task priority > background tasks, < critical ISRs.

43) How do you debug stack overflows in RTOS tasks?

Answer:

- Enable FreeRTOS stack overflow hook (vApplicationStackOverflowHook).
- Fill task stacks with magic value (0xA5A5A5A5) at creation.
- Periodically scan for corruption.
- Use linker map and analyze actual peak usage.

44) How can FreeRTOS task switching impact timing-sensitive bare-metal drivers?

Answer:

- Preemption may delay ISR response or break timing.
- Critical sections (`taskENTER_CRITICAL`) may block ISRs.
  Mitigation:
- Use ISR-safe APIs (`FromISR()` variants).
- Isolate real-time drivers from RTOS using dedicated interrupts or DMA.

45) How do you create a custom section for calibration data in linker script?

Answer:

LD snippet:

```
.cal_data 0x08010000 :
{
  KEEP(*(.cal_data))
} > FLASH
```

C Code:

```
__attribute__((section(".cal_data")))
const uint32_t calib_data[] = { ... };
```

46) How do you reserve a fixed block of RAM for a bootloader or external debugger?

Answer:

- In linker script:

```
MEMORY
{
  BOOT_RAM (rwx) : ORIGIN = 0x20000000, LENGTH = 4K
  APP_RAM (rwx)  : ORIGIN = 0x20001000, LENGTH = 60K
}
```

- Assign .noinit or .bootbuffer section to BOOT_RAM.

47) What is a weak symbol, and how is it useful in embedded startup code?

Answer:

A weak symbol can be overridden by a strong definition.
Use it for handlers (`e.g., __attribute__((weak)) void SysTick_Handler() {}`), so the user can override in application without modifying core files.

48) What are vector table relocation techniques and use cases?

Answer:

- Relocate vector table using `SCB->VTOR` (on Cortex-M3+).
- Common use cases:
  - Bootloader → application jump.
  - RAM-based ISR remapping for dynamic updates.
  - Secure/Non-secure switching (TrustZone).

49) How do you detect UART framing errors in hardware?

Answer:

- Check status flag (e.g., `USARTx->ISR` & `USART_ISR_FE`).
- Occurs if stop bit is missing or invalid.
- Clear by reading data register (`USARTx->RDR`).

50) What's the difference between SPI full-duplex and half-duplex?

Answer:

- **Full-duplex:** simultaneous read/write (MOSI & MISO active).
- **Half-duplex:** shared line (only MOSI or MISO used at a time).
  Use full-duplex for streaming; use half for bidirectional command-response.

51) How do you implement I2C clock stretching handling?

Answer:

- Slave may hold SCL low to delay master.
- Master must monitor SCL and wait before proceeding.
- Most hardware I2C masters (e.g., STM32) support it natively.

52) How do you handle CAN message loss due to buffer overflow?

Answer:

- Enable FIFO full/overrun interrupt.
- Use hardware filters to reduce load.
- Offload to DMA or defer parsing to a high-priority task.

53) How do you detect and recover from I2C bus lockup?

Answer:

- Clock out 9 dummy pulses if SDA stuck low.
- Re-init I2C peripheral.
- Use GPIO mode temporarily to release bus.