Please refer to the below study pointer kindly refer to this. Thank you

---

1) Embedded C Programming -> Pointers, memory Management, Structures, Linked List ISR, bitwise ops, data types

2) RTOS Concepts -> Tasks, Mutex, Semaphores, Spinlock Queues, Scheduling, Process Management, Memory Management

3) Bootloader/U-Boot ->BootROM/ATF, Boot stages, porting, environment, device tree

4) BSP/Drivers -> Peripheral init, register-level debug, HAL

5) Linux Kernel & Device Tree -> Kernel build, DTS, module loading, debugging

6) ARM Architecture -> Exception levels, MMU, caches, TrustZone basics,BBB - sitara

7) Protocol Knowledge -> I2C, SPI, UART, CAN, USB, PCIe

8) Debugging Tools -> JTAG, GDB, oscilloscope, logs, crash analysis

9) Testing/Validation -> Manual testing, scripting, automation (Python/Shell)

10) Build Systems -> Yocto, Buildroot, Makefile, CMake

1️⃣ Embedded C Programming

- **Pointers** → function pointers, pointer to array, pointer to structures, volatile pointers in HW registers.
- **Memory management** → stack vs heap, malloc/free, fragmentation issues.
- **Structures & unions** → padding, packing, bitfields.
- **Linked lists** → circular list, double list (Linux kernel list.h).

- **ISR coding** → re-entrancy, volatile usage, shared data protection.
- **Bitwise ops** → masking, shifting, clearing/setting flags.
- **Data types** → int vs long (16/32/64-bit), endian issues.
- **const vs volatile vs static** keywords usage in embedded.
- **Optimization** → inline, restrict keyword.
- **Common bug** → uninitialized pointers, memory leaks, race conditions.

2️⃣ OS Concepts

- **Tasks/threads** → stack per task, context switch mechanism.
- **Mutex** → binary vs recursive, priority inheritance.
- **Semaphore** → binary vs counting, producer-consumer usage.
- **Spinlock** → difference from mutex (busy waiting, used in ISR context).
- **Queues** → inter-task comms, message passing.
- **Scheduling** → round robin, rate-monotonic, EDF.
- **Priority inversion** problem + solution.
- **Process management** → task states (ready, running, blocked).
- **Memory management** → static vs dynamic allocation in RTOS.
- **OS tick** → SysTick timer, tickless operation for power save.

3️⃣ Bootloader / U-Boot

- **BootROM** → first code in SoC (stored in mask ROM).
- **ATF (ARM Trusted Firmware)** → EL3 secure monitor before kernel.
- **Boot stages** → SPL (MLO) → U-Boot proper → Kernel.
- **Porting U-Boot** → board.c, init sequence, low-level drivers.
- **Environment variables** → bootargs, bootcmd, netboot.
- **Device Tree handoff** → U-Boot passes DTB to kernel.
- **Secure boot** → signed image validation.
- **Peripheral init** → DDR, clocks, pinmux in U-Boot.
- **Recovery modes** → UART boot, USB boot.
- **Fast boot/TFTP boot** → development flows.

4️⃣ BSP / Drivers

- **BSP contents** → bootloader + kernel + drivers + file system.
- **Peripheral init** → clock, reset, GPIO, muxing.
- **Register-level debug** → datasheet + memory map.
- **HAL layer** → abstraction for portability.
- **Driver stacks** → char, block, network drivers.
- **Interrupt handling** → request_irq (), ISR, threaded IRQ.

- **Power management** → suspend/resume callbacks.
- **Platform data vs Device Tree** in driver binding.
- **DMA integration** in drivers.
- **Testing driver** → using sysfs, debugfs, ioctl.

5️⃣ Linux Kernel & Device Tree

- **Kernel build system** → config, defconfig, menuconfig.
- **Device tree (DTS)** → describes HW, not hardcoded in kernel.
- **DTS binding** → compatible strings, reg, interrupts, clocks.
- **Kernel modules** → insmod/rmmod, init/exit functions.
- **Sysfs/procfs/debugfs** → user-space interfaces.
- **Kthreads & workqueues**.
- **Spinlocks, seqlocks** in kernel context.
- **Debugging tools** → printk, dmesg, ftrace, kgdb.
- **Cross compilation** → ARM toolchain.
- **Kernel panic/crash dump** analysis.

6️⃣ ARM Architecture (Sitara AM335x – BBB Example)

- **Exception levels (EL0–EL3)** → user, kernel, hypervisor, secure monitor.

- **MMU** → virtual→physical mapping, page tables.
- **Caches** → data vs instruction cache, coherence.
- **TrustZone** → Secure vs Non-secure world.
- **ARM pipeline** → superscalar, branch prediction.
- **Instruction sets** → ARM, Thumb, NEON SIMD.
- **Interrupt controller** → GIC on Cortex-A.
- **Memory barriers** → DMB, DSB, ISB.
- **Context switch mechanism** in ARM.
- **Sitara SoC internals** → PRU, peripherals, DDR controller.

7️⃣ Protocol Knowledge

- **I2C** → addressing, ACK/NACK, multi-master arbitration.
- **SPI** → CPOL/CPHA, full-duplex, DMA usage.
- **UART** → baud rate calculation, parity, flow control (RTS/CTS).
- **CAN** → message ID arbitration, error handling, CAN-FD.
- **USB** → endpoints, control/bulk/iso transfers.
- **PCIe** → lanes, enumeration, BARs.
- **Ethernet basics** → MAC, PHY, MII/RGMII interface.
- **Debugging** → protocol analyzers, logic analyzers.

- **Error recovery** → retries, bus reset.
- **Throughput vs latency tradeoffs**.

8️⃣ Debugging Tools

- **JTAG/SWD** → halt CPU, step instructions, boundary scan.
- **GDB** → remote debugging, breakpoints, core dump analysis.
- **Oscilloscope** → check waveforms, jitter.
- **Logic analyzer** → protocol decode (I2C/SPI/UART).
- **Kernel logs** → dmesg, printk log levels.
- **Crash analysis** → oops, panic, stack trace decode.
- **Valgrind** → memory leaks (on Linux).
- **Perf/ftrace** → performance profiling.
- **ETM/ETB (ARM trace)**.
- **Bus analyzers** for PCIe/USB.

9️⃣ Testing / Validation

- **Manual testing** → checklist based.
- **Black-box vs white-box** testing.
- **Boundary conditions** → overflow, underflow.

- **Stress testing** → max load, long duration.
- **Scripting** → Python, Shell for automation.
- **CI/CD pipelines** → Jenkins, GitLab CI.
- **Hardware-in-Loop (HIL)** setups.
- **Regression testing** → avoid old bugs return.
- **Code coverage analysis**.
- **Static analysis tools** → cppcheck, Coverity.

🔟 Build Systems

- **Makefile** → dependencies, rules, variables.
- **CMake** → cross-platform build config.
- **Yocto Project** → layers, recipes, bitbake.
- **Buildroot** → lightweight, easy rootfs generation.
- **Cross compilation** → sysroot, toolchains.
- **Image types** → uImage, zImage, fitImage.
- **Init systems** → SysVinit vs systemd.
- **Rootfs contents** → busybox, glibc/uClibc/musl.
- **Board bring-up** → build + flash + boot test cycle.
- **Debugging builds** → verbose flags, log.do_compile in Yocto.

TI Sitara AM3358

1. CPU Core – Cortex-A8

1. Architecture: ARMv7-A (Application profile).
2. Pipeline: Superscalar, dual-issue, 13-stage pipeline.
3. Frequency: Up to 1 GHz.

Caches:

1. 32 KB Instruction + 32 KB Data (L1).
2. 256 KB L2 cache (shared).
3. MMU: Supports virtual memory, required for Linux.
4. NEON SIMD Engine: Vector math, multimedia (FFT, image processing).
5. VFPv3: Hardware floating-point.
6. TrustZone: Secure vs Non-secure world separation (used for security/TEE).

Interviewer trap: "Why not use Cortex-M instead?"

→ *Cortex-A8 supports Linux (needs MMU), while Cortex-M cannot run full Linux.*

2. PRU (Programmable Real-time Unit)

1. Two 32-bit RISC microcontrollers inside SoC (200 MHz each).
2. Independent from Cortex-A8.
3. Direct access to system memory and I/O pins.
4. Ultra-low latency (\~5 ns I/O response).

Used for:

1. Industrial protocols (Profibus, EtherCAT, CAN extensions).
2. Bit-banging custom serial protocols.
3. Motor control and sensor interfacing.

In interviews: mention PRU offloads real-time work while Cortex-A8 runs Linux.

3. Memory & Storage of sitara?

External DDR2/DDR3 interface (BBB uses 512 MB DDR3L).

On-chip:

1. 64 KB SRAM (fast access).
2. 176 KB ROM (BootROM).
3. Boot sources: eMMC, SD card, UART, USB, NAND, NOR.
4. BootROM reads SYSBOOT pins to decide boot sequence.

4. Boot Process of sitara?

1. Power ON → BootROM executes (fixed inside chip).

2. BootROM checks boot mode pins → selects boot medium (e.g., eMMC/SD).

3. Loads MLO/SPL (Secondary Program Loader) from storage into SRAM.

4. SPL initializes DDR → loads U-Boot into DDR.

5. U-Boot initializes peripherals, loads Linux Kernel + Device Tree + initramfs.

6. Kernel mounts root filesystem, runs init/systemd.

In BBB, default boot order: ***eMMC → SD → UART → USB***.

5. Sitara Interconnect & Peripherals

Interconnect: L3/L4 interconnect buses.

L3 (high-speed) connects Cortex-A8, DDR, GPU.

L4 (peripheral bus) connects UART, I2C, SPI, GPIO, etc.

Peripherals supported:

1. 6× UARTs
2. 3× I²C
3. 2× SPI
4. 2× CAN
5. USB 2.0 OTG + Host
6. Gigabit Ethernet MAC (with 2-port switch inside SoC)
7. 8× 12-bit ADC channels
8. PWM, eCAP, eQEP for motor control
9. MMC/SD/SDIO

6. Graphics on Sitara?

1. Integrated Powervar SGX530 GPU.
2. Supports OpenGL ES 2.0.
3. Used in HMI/GUI devices.
4. On BBB: used for HDMI output.

7. Clocking & Power

1. Power domains:
2. MPU (CPU)
3. PER (Peripherals)
4. CORE (DDR, GPU, interconnect)
5. Clock sources: 24 MHz crystal + PLLs.
6. Supports DVFS (Dynamic Voltage & Frequency Scaling).
7. Low power modes: sleep, deep sleep.

8. Linux Integration ,Why Cortex-A8 is good for Linux?

1. Has MMU → required for process isolation & virtual memory.

2. Supports caches + NEON → improves performance.
3. Linux runs at EL1 (privileged mode).
4. Device Tree used to tell Linux about peripherals (pins, clocks, memory).
5. Drivers interact with peripherals via register maps exposed by AM3358.

9. Use Cases of sitara?

1. Industrial automation (EtherCAT, Profibus with PRU).
2. Robotics (motor control + sensor fusion).
3. IoT Gateways.
4. HMI panels with touchscreens.
5. Beagle Bone Black as dev board.

10. Why Sitara AM3358 is Special?

Sitara is Combination of:

1. Application processor (Linux-capable Cortex-A8)
2. Hard real-time PRUs
3. Industrial-grade peripherals (CAN, Ethernet, ADC, PWM).
4. Makes it ideal for embedded Linux + real-time hybrid systems.

Interview Tip (Core Understanding)

If asked “Why AM3358 over others?”:

"Because it provides both a Linux-capable Cortex-A8 core and PRUs for deterministic real-time tasks in the same chip. This hybrid makes it suitable for industrial-grade embedded systems where both high-level OS and real-time precision are needed."

Sitara AM3358 – Core Overview

1️⃣ Core Overview

- Processor Type: ARM Cortex-A8, 32-bit RISC (ARMv7-A).
- Clock: Up to 1 GHz.
- Pipeline: 13-stage, superscalar, dual-issue → executes 2 instructions per cycle.
- Caches:
    - L1: 32 KB Instruction + 32 KB Data
    - L2: 256 KB unified cache
- VFPv3 & NEON SIMD: Hardware floating point + vector acceleration (DSP-style operations).
- TrustZone: Secure vs non-secure execution → useful in secure industrial systems.

Interview Tip: Always mention Cortex-A8 can run Linux because it has MMU, unlike Cortex-M.

2️⃣ PRU – Real-Time Accelerator

- 2× 32-bit RISC cores, 200 MHz each.
- Independent from Cortex-A8, can access GPIO and memory directly.
- Handles deterministic real-time tasks:
    - Bit-banging protocols (custom SPI, I2C)
    - Motor control, PWM, encoder capture
    - Industrial protocols (EtherCAT, CAN extensions)
- Why PRU is unique: Cortex-A8 cannot guarantee sub-microsecond timing, PRU can.

Interview POV: "PRU offloads real-time deterministic tasks from main Linux core, ensuring reliable timing without affecting high-level OS."

3️⃣ Memory & Storage

- External DDR3L SDRAM: 512 MB on BBB → for Linux Kernel + RAM apps.
- On-chip SRAM: 64 KB → used by BootROM/SPL for early boot.
- BootROM (mask ROM inside chip) decides boot source: eMMC, SD, UART, USB.
- Key Concept: Sitara is a microprocessor → needs external memory; MCU has everything on-chip.

4️⃣ Peripherals (Industrial-Grade)

- Communication: 6 UART, 3 I2C, 2 SPI, 2 CAN, USB 2.0 OTG, Ethernet MAC (2-port switch).
- ADC: 12-bit, 8 channels → sensor interfacing.
- PWM/eCAP/eQEP → motor and encoder support.
- GPIO: 90+ multiplexed pins → expansion via capes on BBB.
- Interconnect: L3 (high-speed) → CPU, DDR, GPU; L4 (peripheral bus) → UART, SPI, GPIO, ADC.

Interview POV: "Sitara combines high-level processing and MCU-class I/O on a single SoC — this reduces board complexity and improves reliability."

5️⃣ Graphics (Optional but impressive in interview)

- PowerVR SGX530 GPU → OpenGL ES 2.0
- Handles UI acceleration, used for HDMI output / HMI panels
- Offloads CPU from GUI rendering

6️⃣ Real-Time & Linux Interaction

- Linux kernel runs on Cortex-A8 → EL1 (privileged mode)
- PRU runs independently → deterministic response
- Linux communicates with PRU via shared memory + interrupts
- Device Tree describes: peripherals, pin muxing, clocks, PRU memory mapping

Interview POV: "Hybrid architecture allows Sitara to run full Linux applications while simultaneously handling real-time operations deterministically through PRUs."

7️⃣ Power & Clocking

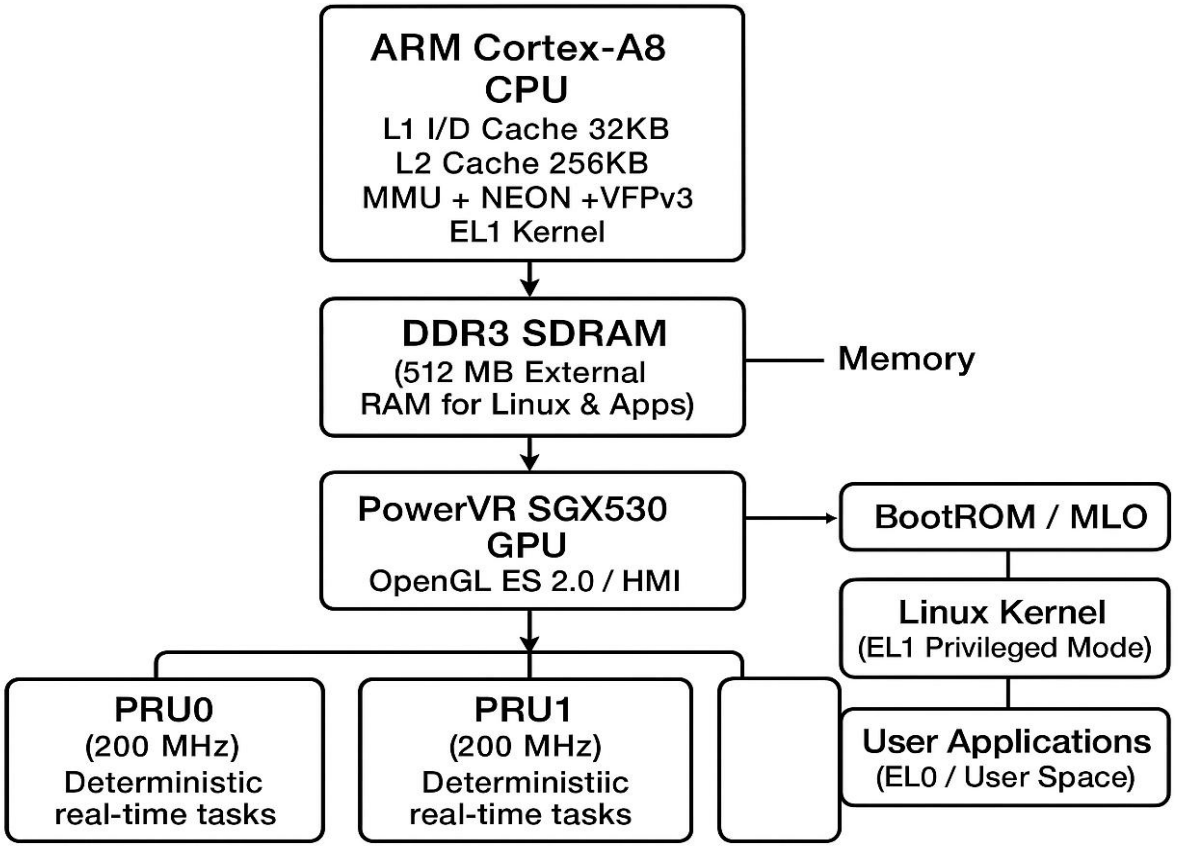
- Power domains: MPU (CPU), PER (peripherals), CORE (DDR/GPU)
- Dynamic Voltage and Frequency Scaling (DVFS) supported → reduces power consumption
- Low-power modes: idle, standby, deep sleep

Interview POV: "AM3358 is optimized for industrial embedded systems where power and performance scaling is required."

8️⃣ Why Sitara AM3358 is Special

- Full-featured Microprocessor (Linux capable) + PRU real-time cores
- Rich MCU-style peripherals integrated
- Supports industrial interfaces (CAN, PWM, ADC, Ethernet)
- Trusted for HMI, industrial automation, robotics, IoT gateways
- Board example: Beagle Bone Black uses AM3358 → shows real-world adoption

Perfect interview one-liner summary:
"AM3358 is a hybrid microprocessor that combines a Linux-capable Cortex-A8 core, deterministic PRU cores, and rich MCU-style peripherals, making it ideal for embedded Linux systems with real-time industrial requirements."



1)Sitara AM3358 – Microprocessor or Microcontroller?

- Sitara AM3358 is a Microprocessor (Application Processor).
    - It is based on ARM Cortex-A8 (ARMv7-A) core, which is a processor-class core (runs Linux/Android).
    - Needs external DDR (SDRAM) for program/data memory → unlike microcontrollers which have flash/SRAM built in.

- Has MMU (Memory Management Unit) → required for Linux and multitasking OS.
- Supports caches (L1 + L2), NEON, TrustZone → features of processors.

◆ Why not Microcontroller?

- MCUs (like Cortex-M, PIC, MSP430, etc.) have:
  - On-chip Flash for code.
  - On-chip SRAM.
  - Limited MHz range (tens to few hundred MHz).
  - No MMU (so can't run full Linux, only bare-metal or RTOS).

But Sitara:

- Has only small on-chip SRAM (64 KB).
- Code runs from external DDR/eMMC/SD.
- Full OS support (Linux, Android, QNX).
- Higher performance (up to 1 GHz).

◆ Why confusion happens?

Because AM3358 also has MCU-like peripherals (UART, SPI, I²C, ADC, PWM, CAN, etc.).
And even has PRU (Programmable Real-time Unit) → looks like a microcontroller *inside*.

So, it is correct to say:

Sitara is an Application Processor (Microprocessor), but with MCU-class peripherals + PRU for real-time.

Perfect Interview Line:
“Sitara AM3358 is a microprocessor based on ARM Cortex-A8, since it requires external memory and runs Linux/OS. But it is special because TI integrated MCU-like peripherals and PRUs inside, so it gives both the flexibility of a processor and the determinism of a microcontroller in one SoC.”

Feature / Aspect	**Microprocessor (MPU)**	**Microcontroller (MCU)**	**Sitara AM3358 (Hybrid)**
Core	Application cores (Cortex-A, x86)	Embedded cores (Cortex-M, AVR, PIC)	ARM Cortex-A8 (Application processor)
Memory	Needs **external DDR/Flash**	Has **on-chip Flash + SRAM**	Needs **external DDR3L (512MB on BBB)**, small 64KB on-chip SRAM
MMU / Virtual Memory	Present → runs Linux/Windows	Absent → only bare-metal or RTOS	Present (MMU) → runs Linux/Android
Clock Speed	500 MHz – GHz	Tens of MHz – few hundred MHz	Up to 1 GHz
Power	High power consumption	Low power (battery devices)	Medium (DVFS, power domains supported)
Peripherals	Limited / external	Rich on-chip (UART, I2C, SPI, ADC, PWM)	Rich MCU-style peripherals + PRU cores
Real-time	Not deterministic	Deterministic (bare-metal/RTOS)	PRU (200 MHz) gives real-time capability
Use Cases	PCs, smartphones, Linux SBCs	IoT nodes, appliances, motor control	Industrial automation, robotics, HMI, BeagleBone Black

Feature / Aspect	Microprocessor (MPU)	Microcontroller (MCU)	Sitara AM3358 (Hybrid)
Examples	Intel i7, ARM Cortex-A53	STM32, PIC32, MSP430	TI AM3358 Sitara

Super Bava 🔥 — this table will **lock your answer in panel round**. You can say it in **flow**, and even draw a quick 3-column comparison on whiteboard.

📌 Microprocessor vs Microcontroller vs Sitara AM3358

Feature / Aspect	Microprocessor (MPU)	Microcontroller (MCU)	Sitara AM3358 (Hybrid)
Core	Application cores (Cortex-A, x86)	Embedded cores (Cortex-M, AVR, PIC)	ARM Cortex-A8 (Application processor)
Memory	Needs **external DDR/Flash**	Has **on-chip Flash + SRAM**	Needs **external DDR3L (512MB on BBB)**, small 64KB on-chip SRAM
MMU / Virtual Memory	Present → runs Linux/Windows	Absent → only bare-metal or RTOS	Present (MMU) → runs Linux/Android
Clock Speed	500 MHz – GHz	Tens of MHz – few hundred MHz	Up to 1 GHz
Power	High power consumption	Low power (battery devices)	Medium (DVFS, power domains supported)
Peripherals	Limited / external	Rich on-chip (UART, I2C, SPI, ADC, PWM)	Rich MCU-style peripherals + PRU cores
Real-time	Not deterministic	Deterministic (bare-metal/RTOS)	PRU (200 MHz) gives real-time capability

Feature / Aspect	Microprocessor (MPU)	Microcontroller (MCU)	Sitara AM3358 (Hybrid)
Use Cases	PCs, smartphones, Linux SBCs	IoT nodes, appliances, motor control	Industrial automation, robotics, HMI, BeagleBone Black
Examples	Intel i7, ARM Cortex-A53	STM32, PIC32, MSP430	TI AM3358 Sitara

---

Interview Takeaway:

Sitara AM3358 is fundamentally a Microprocessor (because of ARM Cortex-A8 + external DDR + MMU + Linux capability).
But it integrates MCU-like peripherals and PRUs → giving best of both worlds.

Final One-liner:
"Sitara AM3358 is a microprocessor with MCU-like features — that's why it powers the Beagle Bone Black as a Linux-capable, real-time friendly SoC."

Flow Explanation – Interview POV

1. CPU (Cortex-A8): Executes Linux kernel + user apps. Uses MMU, L1/L2 cache, NEON/VFP for acceleration.
2. PRU Cores: Handle real-time tasks, independent of CPU, access GPIO, PWM, ADC directly.
3. Memory: External DDR holds Linux + app code/data. Small on-chip SRAM for bootloader/early init.
4. GPU: Offloads graphics/UI tasks from CPU (OpenGL ES 2.0).
5. Peripherals: MCU-style interfaces for sensors, motors, industrial comms.
6. BootROM/MLO: Early initialization, loads U-Boot, sets up DDR.
7. Device Tree: Tells Linux kernel about hardware, pinmux, clocks, PRU memory.

8. Linux Kernel: Runs in EL1, schedules tasks, communicates with PRU, manages peripherals.
9. User Applications: Run in EL0, use drivers to access peripherals.

Key Interview Points to Highlight Verbally:

- Sitara = Microprocessor + MCU peripherals + PRU real-time cores
- Cortex-A8 runs Linux → PRU offloads deterministic tasks
- Device Tree enables OS-hardware binding
- DDR external memory for kernel + apps, small SRAM for boot
- GPU accelerates HMI, reduces CPU load

Below list of Embedded/Firmware C Language Programming Test Questions that assess practical coding skills using brute-force approaches

These are commonly used in interviews C language programming skills:

- Bitwise manipulation

- Pointer logic

- Memory constraints

- Control flow

- Algorithmic thinking without libraries

1. Reverse an Integer (No Library Functions)

Problem: Reverse a given integer.

Brute Force Hint: Use modulus and division.

2. Count Number of 1’s in Binary (Hamming Weight)

Problem: Count set bits in an integer.

Brute Force Hint: Use shifting and masking.

3. Find the Maximum Element in an Array

Problem: Given an array, find the largest number.

Brute Force Hint: Traverse all elements.

4. Check if a Number is Prime

Problem: Check if a number is prime.

Brute Force Hint: Check all numbers up to n-1.

5. Fibonacci Series (Iterative Method)

Problem: Print first N Fibonacci numbers.

Brute Force Hint: Use loop without recursion.

6. Check for Palindrome String (char array)

Problem: Check if string is a palindrome.

Brute Force Hint: Compare start and end characters.

7. Count Occurrences of a Character in a String

Problem: Count how many times a character appears.

Brute Force Hint: Linear scan.

8. Swap Two Numbers Without Temporary Variable

Brute Force Hint: Use arithmetic or bitwise XOR.

9. Array Rotation by K Elements

Problem: Rotate array by K steps.

Brute Force Hint: Use nested loops

10. Find Duplicate in an Array (Brute Force)

Problem: Find any duplicate element.

Hint: Use nested loops.
