Advanced Embedded Systems and C Programming
Interview Questions and Answers ☺

Q1. What is the difference between macro and inline function?

⇒ **Macro (#define)** :- Preprocessor directive that replaces code for compilation.

Inline Function (inline) :- A function where the compiler replaces the function call with the function code to reduce overhead.

Example of Macro (No type checking, Can cause Bugs) :

```
#define SQUARE(x) (x*x)
int result = SQUARE(5+1);  // Expands to (5+1*5+1)
                           = 5+5+1 = 11 (Incorrect)
```

Example of Inline Function (Safe, Type-Checked) :

```
static inline int square (int x) {
   return x*x;
}
int result = square(5+1);  // correct output : 36
```

- Inline functions are better than macros because they are type-safe, debuggable, and avoid side effects.

Q2. What is a Watchdog Timer (WDT) in Embedded System?

⇒ A Watchdog Timer (WDT) is a safety mechanism that reset the system if it detects a software hang or infinite loop.

- It prevents the system from getting stuck indefinitely due to bugs or deadlocks.

- If the WDT is not reset within a specific time, it assumes the system has crashed and perform a reset.

Example of WDT Usage (AVR microcontroller) :

```c
#include <avr/wdt.h>
int main () {
    wdt_enable (WDTO_2S) ;   // set watchdog timeout
                             //   to 2 seconds
    while (1) {
        wdt_reset () ;   // Reset watchdog timer
    }
}
```

Q3. What is a Memory Leak ? How do you prevent it ?

⟹ A memory leak occurs when dynamically allocated memory (malloc()) is not freed, causing memory exhaustion.

- It is a major issue in embedded systems where RAM is limited.

Example of Memory leak (Incorrect code) :

```c
void leak () {
    int *ptr = (int*) malloc (10*sizeof (int)) ;
    // memory allocated but never freed (leak)
}
```

Fixed code (Proper Memory Management) :

```c
void no_leak () {
    int *ptr = (int *) malloc (10*sizeof (int)) ;
    free (ptr) ;   // Free memory after use
}
```

Q4. What is the difference between Polling and Interrupt?

⇒ Polling :- CPU continuosly checks a flag to see if an event has occured. Inefficient and wastes CPU time.

Interrupts :- CPU stops current execution only when an event occurs. More efficient than polling.

Example of Polling (Inefficient) :

```
while (!( UART_STATUS & 0x01)) {

} // Keep checking if data is available.
```

Example of Interrupt (Efficient, Event - Driven) :

```
void _attribute_ ((interrupt)) UART_ISR () {
    char data = UART_Read ();
}.
```

Q5. What is the difference between Blocking and Non-blocking code?

⇒ Blocking Code :- Waits until a task is complete before continuing (e.g., while () loop).

Non-blocking Code :- Perform other tasks while waiting (e.g., interrupt - based handling).

Example of Blocking Code (Bad for Real - Time Systems) :

```
while (UART_BUSY);  // Stalls execution until UART
                    //         is free
```

Example of Non-Blocking Code (Better for Real-Time):

```
if (! UART_BUSY) {
    UART_WRITE (data);
}.
```

Q6. Explain Bit Manipulation and its importance in Embedded Systems.

⇒ Bit manipulation is used to efficiently control hardware registers using bitwise operations.

. It reduce memory usage, increases speed, and optimizes performance.

Example : Set, Clear, Toggle, and check Bit in a Register:

```
#define SET_BIT (PORT, BIT)      (PORT |= (1 << BIT))
#define CLEAR_BIT (PORT, BIT) (PORT &= ~(1 << BIT))
#define TOGGLE_BIT (PORT, BIT) (PORT ^= (1 << BIT))
#define CHECK_BIT (PORT, BIT) (PORT & (1 << BIT))


// Example Usage

SET_BIT (GPIO_PORT, 5); // set bit 5
CLER_BIT (GPIO_PORT, 5); // Clear bit 5
```

Q7. What are the Different types of Embedded system Memory?

⇒ 1) RAM (Random Access Memory) :- Volatile memory for temporary usage.

2) ROM (Read-only Memory) :- Stores firmware; non-volatile.

3) EEPROM (Electrically Erasable Programmable Read-Only Memory) :- Stores settings/configuration, can be rewritten.

4) Flash memory :- Used for program storage (like firmware in microcontroller).

Example :- Flash vs EEPROM Storage in Embedded C :

```
const char firmware_version [] = "v1.2"; // stored in
                                                    flash
EEPROM_Write (0x10, config_data); stored in EEPROM
```

Q8. What is DMA (Direct Memory Access) and why it is importance ?

⇒ DMA allows peripherals to access memory without CPU involvement, improving system performance.

· It reduces CPU overhead and is commonly used for high-speed data transfer in ADC, SPI and UART .

Example : Using DMA for High-Speed Data Transfer :

```
DMA_SetSource (UART_RX_BUFFER);
DMA_SetDestination (RAM_BUFFER);     } Pseudocode
DMA_Enable ();
```

Q9. What is the difference between Harvard and Von Neumann Architectures ?

⇒ **Harvard Architecture :-**

· Separate memory for program (code) and data .

· Faster execution because instructions and data can be fetched simultaneously .

· Used in AVR, ARM Cortex-M microcontrollers .

Von Neumann Architecture :-

· Shared memory for program and data .

· Slower because only one access can occur at a time . //

· Used in General-purpose computers, Embedded linux systems .

Q10. What is the difference between a Bootloader and a Firmware ?

⟹ Bootloader :-

- First piece of code executed at startup.
- Initializes hardware and loads the main application (firmware).
- Used for firmware updates.

Firmware :-

- The main embedded application that runs the device.
- Controls hardware and executes tasks.

Example : Bootloader jumping to Firmware in Embedded C :

```c
void bootloader () {
    if (FIRMWARE_VALID) {
        jump_to_application (); // Execute main firmware
    } else {
        download_firmware (); // Load new firmware
    }
}
```

Final Thoughts :- These Embedded Systems and C interview questions cover advanced memory management, interrupts, bootloaders, bit manipulation, DMA, and real-time processing.

HAPPY LEARNING ☺

Q11. What happens when you dereference a NULL pointer?

⟹ Dereferencing a NULL pointer leads to undefined behaviour and can cause a segmentation fault (crash).

. Since NULL (0x0) does not point to a valid memory location, attempting to access it results in an access violation.

Example (Incorrect Code - Causes Crash):

```
int *ptr = NULL;
printf ("%d", *ptr); // Dereferencing NULL pointer
                     (undefined behaviour)
```

How to prevent it?

. Always check for NULL before dereferencing:

```
if (ptr != NULL) {
    printf ("%d", *ptr);
}
```

Q12. What is the difference between Hard Real-Time and Soft Real-Time Systems?

⟹ Hard Real-Time Systems:

. Strict timing constraints (missed deadlines = system failure).

. Used in automotive, medical devices, avionics (e.g., ABS, pacemakers, flight control).

Software - Real Time Systems:

. Some delay is tolerable; performance degrades but doesn't fails.

. Used in video streaming, VoIP, robotics (e.g., multimedia processing, online gaming).

Example :
- Hard Real-Time :- Airbag deployment system (must trigger within milliseconds).
- Soft Real-Time :- Video buffering (minor delays allowed).

Q13. What is the difference between Mutex and Semaphores in RTOS ?

⇒ Mutex (Mutual Exclusion) :
- Used to protect a shared resource from multiple tasks.
- Only one task can own a mutex at a time.
- Example : Preventing race conditions in multithreading.

Example (Mutex in RTOS - PseudoCode):

```
mutex_lock (mutex) ;   // Task locks resources
   // critical Section
mutex_unlock (mutex) ; // Task releases resources
```

Semaphores :
- Used for synchronization between tasks.
- Can allow multiple tasks to access a resource.
- Example : Managing multiple I/O peripherals.

Example (Semaphore in RTOS - Pseudocode) :

```
semaphore_wait (sema) ; // Task waits for the semaphore
   // Execute task when semaphore is available
semaphore_signal (sema) ; // Release semaphore
```

Q14. Explain Interrupts and Interrupt Latency ?

⇒ • **Interrupt** :- A hardware or software signal that temporarily halts CPU execution to handle higher-priority tasks.

• **Interrupt Latency** :- The time taken by the CPU to respond to an interrupt after it is triggered.

Reducing Interrupt Latency :-

• Optimize Interrupt Service Routine (ISR) execution time.
• Used nested vector interrupt controller (NVIC).
• Disable unnecessary interrupts to avoid delays.

Example : Handling GPIO Interrupt in Embedded C :-

```c
void _attribute_((interrupt)) GPIO_Interrupt_Handler () {
    if ( GPIO_STATUS & 0x01) {   // check interrupt flag
        GPIO_CLEAR_FLAG = 0x01 ;   // clear interrupt flag
    }
}
```

Q15. Explain static and extern keyword in C ?

⇒ **static (storage class modifier)**

• Restricts the variable's scope to the file or function in which it is declared.
• Preserves the value of a variable between function calls.

Example (static variable retains value across function calls) :

```c
void counter () {
    static int count = 0 ; // Retains value across function
    count ++ ;                              calls
    printf ("count : %d \n", count) ;
}
```

Extern (Global Variable Access Across Files)

• Used to declare a variable without defining it.
• Help in sharing variables across multiple files.