Technical Questions on C and Embedded Systems ☺

Q1. Difference between C and Embedded C ?

⇒ C Programming :

- Used for general purpose applications, OS development, and software tools.
- Uses standard libraries (stdio.h, stdlib.h, etc).
- High portability across different hardware platforms.
- No direct access to hardware registers.

Embedded C :

- Designed for microcontrollers and embedded systems.
- Directly interacts with hardware (eg., controlling GPIO, UART, I2C).
- Uses memory-mapped registes instead of standard libraries.
- Optimized for power efficiency and real-time constraints.

Example Code Comparison :-

- C Program (Standard Output) :

```
#include <stdio.h>
int main () {
   printf ("Hello, World! \n");
   return 0 ;
}
```

- Embedded C Program (Microcontroller LED ON/OFF) :-

```
#include < avr/io.h >
int main () {
    DDRB | = ( 1 << PB0) ;  // Set PB0 as output
    PORTB | = ( 1 << PB0) ; // Turn ON LED
    while (1) ;
    return 0 ;
}
```

Q2. Volatile vs Const in Embedded Systems.

⇒ Volatile :-

- Tells the compiler not to optimize the variable.
- Used when a variable's value can change unexpectedly (e.g., hardware registers, interrupt flags).
- Example : Reading a sensor value that may change at any time).

```
volatile int sensor_value ;
void read_sensor () {
    sensor_value = *(volatile int *) 0x40020000;
}
```

Read sensor data

Const :

- Declares a read-only variables whose value cannot change.
- Used for defining fixed values like baud rate, PI etc.
- Examples :

```
const int BAUD_RATE = 9600 ; // Fixed UART baud rate
```

Key Difference :-

- volatile prevents unwanted optimizations.
- Const ensures a variable remains unmodifiable.

Q3. Pointers in Embedded Systems.

⇒ Definition :- A pointer is a variable that stores the memory address of another variable.

Uses in Embedded Systems :-

- Accessing hardware registers.
- Implementing efficient memory management.
- Using function pointers for callback mechanisms.

Example : Direct register Access using Pointers :

```
#define GPIO_PORT *((volatile unsigned int*) 0x40020C00)
void main() {
    GPIO_PORT = 0XFF ; // Turn ON all GPIO pins
}
```

↑ Register Address

Function Pointers in Embedded Systems :-

- Used for ISR (Interrupt Service Routine) handlers.
- Example :

```
void (*reset_func)(void) = (void(*)(void)) 0x00000000 ;
reset_func() ; // jump to reset address
```

↑ Assign reset vector

Q4. Stack vs Heap in Embedded Systems.

⇒ Stack :

- Stores local variables and function call information.
- Memory allocation is automatic (allocated and deallocated by function calls).
- Fast access, but has a limited size.

Example :-

```
int a = 10 ; // stored in stack
```

Heap :

- Stores dynamically allocated memory.
- Memory allocation is manual (needs malloc() and free()).
- More flexible but slower than stack.

Example :-

```
int* ptr = (int*) malloc (sizeof(int)) ;
free(ptr) ;
```

↑ Allocated in Heap

↑ Prevent memory leak.

Issue in Embedded Systems :-

- Stack overflow occurs when recursion is used without a base condition.

Example of stack overflow issue :-

```
void recursive_function () {
    recursive_function () ; // no base case = stack
}                                overflow !
```

Q5. UART, SPI and I2C in Embedded Systems.

⇒ UART (Universal Asynchronous Receiver-Transmitter) :-

- Asynchronous serial communication (no clock signal).
- Uses TX (Transmit) and RX (Receive) lines.
- Common baud rates : 9600, 115200 bps.

Example :-

```
UART_Write ('A') ; // sends character 'A'
```

SPI (Serial Peripheral Interface) :-

- Synchronous serial communication (uses a clock signal)
- Uses 4 wires : MISO, MOSI, SCK, SS.
- Used in SD cards, sensors, and flash memory

Example :-

```
SPI_Write (0xAA) ; // Sending data to SPI device
```

I2C (Inter-Integrated Circuit) :-

- Uses only 2 wires : SDA (Data) and SCL (Clock).
- Supports multiple slave devices.
- Used in temperature sensors, EEPROMs, and RTCs.

Example :-

```
I2C_Start () ;
I2C_Write (0x50) ; // Send device address
```

Q6. CAN Protocol in Automotive ?

⇒ A message based protocol used in automotive and industrial applications.

- Supports multi-master communication.
- Provides error detection and fault tolerance.

Features :-

- Priority-based arbitration (higher priority messages get access first).
- Supports multiple nodes on a single bus.
- Real-time performance (fast response in vehicle control systems)

Example :- CAN message Transmission in C :-

```
CAN_Transmit (messageID, data, length);
```

Q7. RTOS vs General OS :-

⇒ RTOS (Real-Time Operating System) :

- Used in automotive ECUs, industrial automation, and medical devices.
- Provides deterministic task scheduling (guaranteed execution time).
- Low latency for real-time applications.

Example RTOS task creation :-

```
xTaskCreate (Task1, "Task1", 100, NULL, 1, NULL);
```

General OS (Windows, Linux, etc) :-

- Used for general computing tasks.
- Non-deterministic scheduling (task execution order is unpredictable).
- Designed for user applications like web-browsing, gaming and office work.

Q8. Debugging tools in Embedded Systems.

⇒ · GDB (GNU Debugger) :- Used for debugging C programs in Linux.
· CANoe :- Used for testing and debugging CAN protocol based application.
· SonarQube :- Used for code quality analysis and bug detection.

Q9. Code optimization in Embedded Systems :-

⇒ · Use Bitwise Operation instead of Arithmetic :

```
#define SET_BIT (PORT, BIT) (PORT |= (1<< BIT))
```

⇑ Faster than arithmetic

· Reduce function calls and loops to save execution time.
· Use static, const to optimize memory usage.
· Enable Compiler optimization flags (-O2, -O3).

Final Thoughts :- These are the technical concepts you must master before your Embedded Developer or C Developer interview.

Q10. Difference between macro and inline function ?

⇒ Features	Macros	Inline function
1) Definition	`#define SQUARE(x) (x*x).`	`static inline int square(int x) { return x * x; }`
2) Type Safety	No type checking.	Type-safe.
3) Debugging	Hard to debug.	Easier to debug.
4) Performance	Faster (avoids function call overhead).	Slightly slower but safer.

General Interview Questions for Embedded Dev ☺

Q1. Tell me about yourself.

⇒ I am an Electronics and Communication Engineer specializing in Embedded Systems, with expertise in C, C++, Embedded C, and Linux OS. I have hands-on experience in firmware deployment, debugging, and working with communication protocols like UART, SPI, CAN and I2C. In my role at current organization, I contributed to developing camera communication modules, reducing latency, and optimizing embedded systems. I have a also worked on AI/ML integration, annotation tools and vision deployment. I am passionate about solving complex technical challenges and developing optimized Embedded Solutions.

Q2. Why do you want to switch jobs?

⇒ I am looking for a new opportunity where I can further enhance my Embedded systems skills, take on more challenging projects, and contribute to innovative solutions. While I have gained valuable experience in my current role, I believe that this opportunity will allow me to grow both technically and professionally.

Q3. What are your strength and weakness?

⇒ Strengths:

- Strong programming skills in C and Embedded C.
- Experience with Linux OS and firmware deployment.
- Knowledge of communication protocols (UART, SPI, CAN).
- Quick problem-solving and debugging skills.
- Experience working in automotive and embedded domain.

Weakness:

- Sometimes I get too focused on optimizing code and spend extra time on minor improvements. However, I have learned to balance optimization with deadlines.

Q4. What do you knows about our company ?

⇒ Research the company's products, services, and recent projects before the interview. Mention any embedded projects they are working on and how your skills allign with their requirement.

HAPPY LEARNING ☺