Assignment-1

1. What is the difference between General purpose system and embedded system?

Sol: → what is general purpose system

→ what is embedded system

→ General purpose sim v/s embedded system

* A General purpose computer is one that given the appropriate application and required time, should be able to perform most common computing tasks.
* personal computers, including desktops, laptops, notebooks are all examples of general purpose computers

Embedded Systems-

* An embedded system is a computer system with a dedicated function within a larger system.

Ex:- calculator.

* It consists of both * Hardware
  * software

Parameters	General purpose computer	Embedded system
1. purpose	* multipurpose	* single functioned
2. size of system	* Big	* small
3. power consume	* more	* less
4. cost of system	* costly	* cheap
5. memory	* Higher memory	* lower memory
6. performance	* fast & better performance	* fixed runtime required

7) User interfaces * keyboard, display, mouse, touchscreen * Button, sensors (gas, IR)

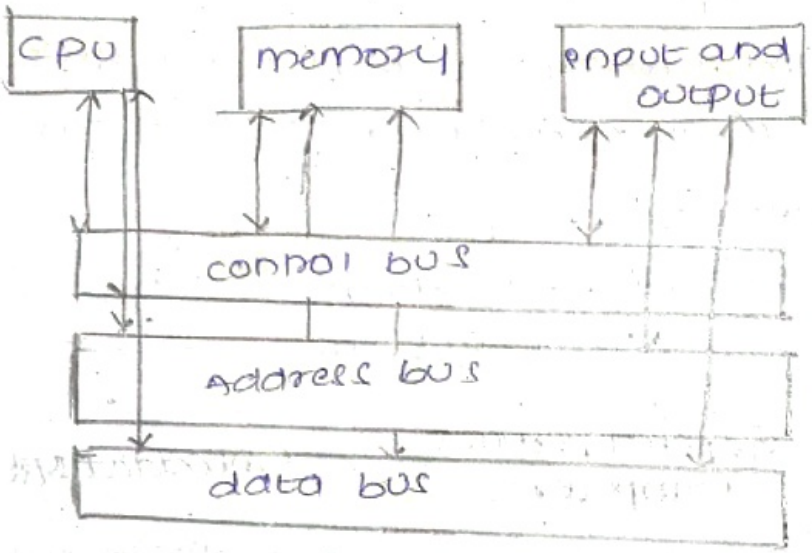
2. What are device drivers?

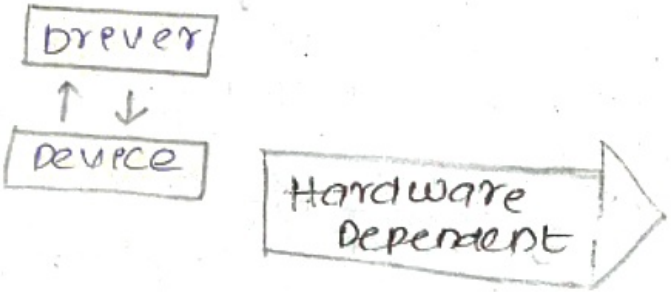
Sol: In computing, a device driver is a computer program that operates or controls a particular type of device that is attached to a computer.

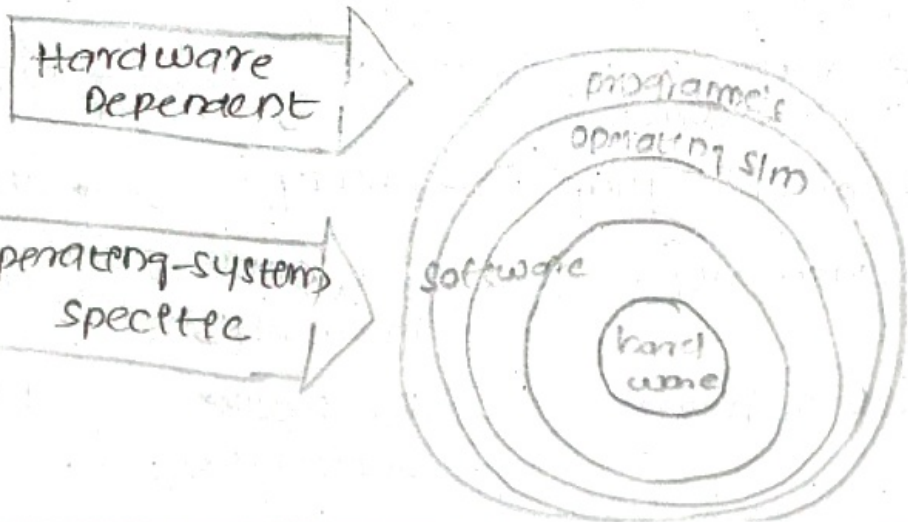
Applications & user

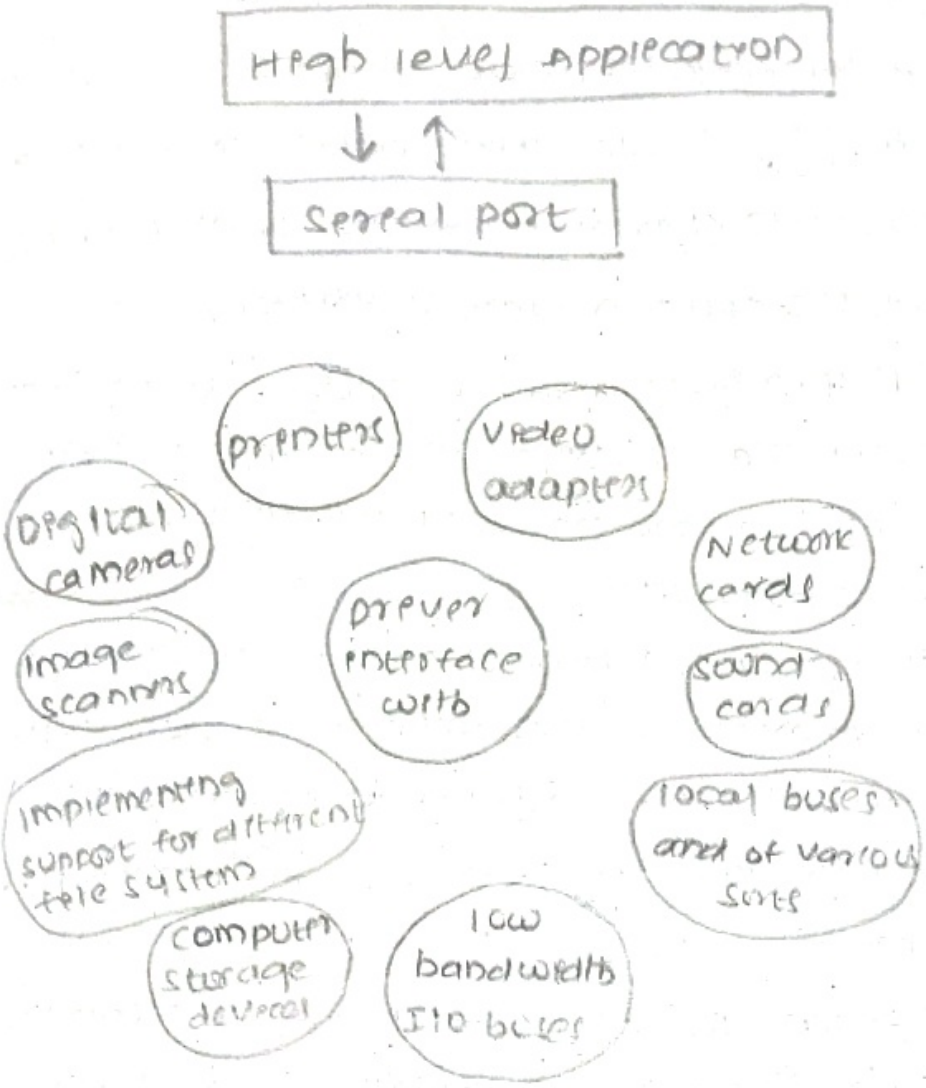
operating system

device drivers









3. How can hardware understand code that we written the embedded system?

Sol: 1. Processor (CPU): The CPU is the brain of the embedded system. It executes instructions sequentially. When you write code, it's eventually compiled into machine code (binary instructions) that the CPU can understand. These instructions tell the CPU what operations to perform.

2. Memory:- Embedded systems have both program memory (where the code is stored) and data memory (for variables and runtime data). The CPU fetches instructions from the program memory and operates on data in the data memory.

3. Compilation: You write code in a high-level programming language like C or C++. The code is then compiled using a compiler that translates your code into machine code (binary instructions). This machine code is what the CPU can execute directly.

4. Linking: If your code involves multiple source files or libraries, a linker combines all the compiled code into one binary file that can be loaded into memory and executed by the CPU.

5. Loading: The compiled binary code is loaded into the program memory of the embedded system, often through mechanisms like flashing or burning it into non-volatile memory (e.g., flash memory or EEPROM).

6. Execution: The CPU fetches and executes instructions one by one from the program memory. It performs operations like arithmetic calculations, data manipulation, and control flow based on these instructions.

7. I/O Interfaces: The embedded system interacts with the external world through I/O interfaces like GPIO (General Purpose Input/Output), UART (Universal Asynchronous Receiver/Transmitter), SPI (Serial Peripheral Interface), and others. Your code includes instructions to read from write to these interfaces, allowing the hardware to communicate with sensors, actuators, displays, and other peripherals.

In summary, hardware in embedded systems understands code through the translation of high-level code into machine code, which is executed by

the cpu. The code interacts with memory and I/O interfaces to perform specific tasks and control external devices based on your program's logic.

4. what is the difference between OS and RTOS?

Sol:-

characteristics	RTOS	OS
complexity	Lightweight and designed for minimal overhead and reduced complexity	more complex, supporting a wide variety of applications and hardware configuration
Application domain	commonly used in domains where timing and determinism are critical, such as aerospace, automotive, industrial control systems, medical devices and robotics.	suitable for a broader range of applications, including desktop computers, servers, mobile devices, and consumer electronics.
Examples	FreeRTOS, VxWorks, QNX, eCos, and RT-Linux.	Windows, macOS, Linux, and Unix variants.
Resource utilization	optimized for minimal resource usage and efficient memory management.	optimized for efficient resource utilization and user experience
cost	High cost	Low cost

Determinism	Deterministic execution with guaranteed timing and deadlines	Non-deterministic execution with no guarantees of timing or deadlines
Real-time behaviour	provides deterministic behaviour	primarily focuses on multitasking and resource sharing.