Q1. Draw the System block diagram of the Embedded System made by you in the project? Please explain different blocks.

Ans: **It can vary depending on your project**.

A general block diagram for an embedded system.

1. Microcontroller/Processor: This is the heart of the embedded system, responsible for executing the program instructions and controlling the overall system.

2. Input/Output (I/O) Interfaces: These interfaces facilitate communication between the microcontroller and external devices. They can include digital I/O pins, analog-to-digital converters (ADCs), digital-to-analog converters (DACs), serial communication ports (UART, SPI, I2C), etc.

3. Sensors: These devices gather data from the system's environment. Examples include temperature sensors, humidity sensors, accelerometers, gyroscopes, etc.

4. Actuators: These devices convert electrical signals into physical actions. Examples include motors, relays, solenoids, LEDs, etc.

5. Power Supply: This block provides the necessary power to the system. It may include voltage regulators, batteries, or external power sources. Would be good to show if this is battery operated design ? Capacity of Battery or voltage etc mentioned shall be good.

6. Communication Interfaces: These interfaces enable communication between the embedded system and external devices or networks. Examples include Ethernet, Wi-Fi, Bluetooth, Zigbee, etc.

7. User Interface: This can be any user's interface with the system. It may include buttons, switches, displays (LCD, LED, OLED), keypads, touchscreens, etc.

8. Real-Time Clock (RTC): RTC provides accurate timekeeping functionality to the system, useful for applications requiring scheduling or time-sensitive operations.

9. Peripheral Modules: These are additional modules or components specific to the application's needs. They can include motor drivers, communication modules (GSM, GPS), SD card interfaces, etc.

10. System Software: This includes the embedded firmware or operating system running on the microcontroller. It controls the behaviour of the system and manages hardware resources.

11. Application Software: This user-facing software runs on top of the system software. It implements the specific functionality or application logic desired for the embedded system.

These are the basic blocks you'll often find in an embedded system. The actual layout and components may vary depending on the specific requirements and complexity of the system.

Q2. Write down 5 technical challenges you encountered during the development of the project. How did you solve them? Explain with specifics.

Ans: **It can vary depending on your project**.
Would be good to see Challenges and their solutions. The explanation shall be technical with right kind of information.
Writing soldering issues and loose connections etc shall be explained with what it caused and how the permanent solution was done.

Q3. Did you do the power analysis of the project? What are the key findings of the power analysis in your project? How did it impact your design choices?

Ans: **It can vary depending on your project**.

a. Expected outcome is to see if you mentioned different voltages and current consumption in each block.
b. Expected outcome is to see if you mentioned approx running of the design on the battery if the design is battery operated ?
c. Additionally, if the design was using external adaptors or power supply, the current consumption was mentioned ?
d. Any low-power modes mentioned ?

1. Key Findings:

- Power Consumption: Identifying the system's overall power consumption under different operating conditions and states (active, sleep, standby) is crucial. This includes analyzing power usage by individual components such as the microcontroller, sensors, actuators, communication modules, etc.

- Power Profiles: Understanding the power profiles of different operations or tasks within the system helps pinpoint areas where optimization can yield significant power savings.

- Peak Power Demands: Identifying peak power demands during certain operations or events helps dimension power supply requirements and manage power distribution.

- **Battery Life Estimation:** Estimating the system's battery life under various usage scenarios is essential for battery-powered applications. It involves calculating average and worst-case power consumption to determine battery size and optimize power-saving techniques.

- **Heat Dissipation:** High power consumption can lead to heat generation, affecting system reliability and longevity. Analyzing heat dissipation requirements ensures proper thermal management in the design.

2. Impact on Design Choices:

- **Component Selection:** Power analysis influences the selection of components with low power consumption characteristics, such as microcontrollers with low-power modes, efficient sensors, and optimized communication modules.

- **Software Optimization:** Understanding power consumption profiles guide software optimization efforts, such as minimizing processor wake-up time, optimizing algorithms for energy efficiency, and implementing sleep modes during idle periods.

- **Power Supply Design:** Power analysis helps dimension the power supply unit (PSU) appropriately to meet peak power demands and ensure stable operation under all conditions. It may also influence the choice of voltage regulators and power management ICs.

- **Energy Harvesting:** For energy-harvesting applications, power analysis informs the selection of suitable energy sources (solar, kinetic, thermal) and the design of energy harvesting circuits to maximise energy conversion efficiency.

- **System Architecture:** Power analysis may lead to changes in the system architecture, such as distributed power management, dynamic voltage scaling, or partitioning tasks to optimize power consumption in different system components.

Overall, power analysis plays a crucial role in designing energy-efficient embedded systems, impacting component selection, software optimization, power supply design, and overall system architecture.

Q4. Which project apart from yours was interesting for you from the design done by other groups ? Which key features of the project were interesting for you in terms of ESD course content? Please explain in atleast 4 comments.

Ans: **It can vary depending on your choice of project**.
Expected outcome includes if you were attentive to other team members presentations ?
Did you pick-up any learnings from the team ?
How did you fit it into ESD course content learnings:

a. MCU used in the design ?
b. Algorithm challenge
c. Identification of components used in the design and approx block diagram
d. Any communication interface used
e. Any Software developed on App/ Web browser mentioned ?

Q5. Which evaluation board have you used in your project?

Ans: **It can vary depending on your choice of project**.

Expected outcomes:

a. Generally Arduino is based on MCU which is powered by 5V. Other designs have MCUs which are working at 3.3V
b. Frequency at which MCU is working is generally MHz.

Features:

a. Would expected communication interface available on the board
b. Any user interface available for the users
c. Any GPIO being used are mentioned
d. Any debug interface mentioned
e. Any development IDE or ease of debug mentioned

Q 6. What are the decision-making factors between the usage of a professional (license paid) or open-source toolchain that a product design engineer has to take? Justify with atleast 4 factors and examples in tabular form.

Ans: When deciding between a professional (licensed) toolchain and an open-source toolchain, product design engineers must consider several factors to make an informed decision. Here are four key factors, along with examples:

Decision Factor	Professional Toolchain	Open-Source Toolchain
Cost	Requires upfront licensing fees and ongoing maintenance costs.	Typically free to use, reducing financial barriers.
Example	Autodesk AutoCAD, SolidWorks, Altium Designer	Blender, FreeCAD, KiCad
Support & Documentation	Often comes with comprehensive customer support, extensive documentation, and training resources.	Relies on community support, documentation may vary in quality.
Example	ANSYS, Siemens NX, MATLAB	GNU Octave, OpenFOAM, OpenCV
Customization & Flexibility	Limited customization options, but tailored for specific industries and standards.	Highly customizable, allowing users to modify and extend functionalities.
Example	CATIA, Cadence Allegro, PTC Creo	OpenSCAD, OpenBOM, Arduino
Ecosystem & Integration	Offers seamless integration with other professional tools and software suites.	Integration may require additional effort due to varied compatibility.
Example	Adobe Creative Suite, Altair HyperWorks, PTC Windchill	GIMP, Inkscape, LibreOffice

Q7. In a smart-street lighting system, which multiple communication technologies can be used between the light controller and the web interface? Justify with reasons for such choices.

Ans: In a smart street lighting system, multiple communication technologies can be used between the light controller and the web interface to enable remote monitoring, control, and management of the street lights. Here are different communication technologies along with their justifications:

— High Priority —-

1. Cellular (3G/4G/5G):

- Justification: Cellular communication provides reliable connectivity over long distances, making it suitable for remote monitoring and controlling street lights in areas without Wi-Fi coverage. It allows street lights to be monitored and controlled from anywhere with cellular network coverage. Additionally, cellular networks offer high data throughput, enabling the transmission of large amounts of data such as sensor readings and status updates.

2. LoRaWAN:

- Justification: LoRaWAN is a low-power, wide-area network (LPWAN) technology that offers long-range communication with low power consumption. It is well-suited for smart street lighting systems deployed over a wide area, as it allows for the communication of sensor data and control commands over long distances without the need for frequent battery replacements. LoRaWAN networks can be deployed using gateway nodes to cover large geographical areas.

3. Zigbee:

- Justification: Zigbee is a wireless mesh networking technology that offers low-power communication between street lights and the central management system. It is suitable for forming self-organizing networks of street lights, allowing them to communicate with each other and relay data back to the web interface. Zigbee networks can be easily expanded and scaled to accommodate additional street lights and devices.

4. Sub-Giga Ghz connectivity

Communication like 865-868MHz can be used for making Local Mesh network which can make network of the Smart-street lights. These data can be taken to the cloud using a Gateway based on WAN technology.

— Low Possibility —-

5. Ethernet:

- Justification: Ethernet provides reliable and high-speed wired connectivity, making it suitable for connecting the light controller to the web interface in locations with existing wired infrastructure. It offers low latency and high bandwidth, ensuring fast and stable communication between the street lights and the central management system. Ethernet is commonly used in smart city deployments where wired connectivity is available.

6. Bluetooth Low Energy (BLE):

- Justification: BLE is ideal for short-range communication between the light controller and nearby devices such as smartphones or tablets. It can be used for local configuration, firmware updates, and troubleshooting of individual street lights. BLE offers low power consumption, making it suitable for battery-operated devices and energy-efficient street lighting systems. BLE is ONLY useful when there is a gateway which was WAN connectivity to take the data to cloud.

7.Wi-Fi:

- Justification: Wi-Fi offers high-speed wireless connectivity, making it suitable for transmitting real-time data between the light controller and the web interface. It provides sufficient bandwidth for streaming video feeds, firmware updates, and configuration changes. Additionally, Wi-Fi networks are widely available in urban areas, making them convenient for street lighting systems.

By using a combination of these communication technologies, smart street lighting systems can achieve reliable and efficient communication between the light controller and the web interface,

enabling remote monitoring, control, and management of the street lights. The choice of communication technology depends on factors such as coverage area, power consumption, data throughput, and existing infrastructure.

Q8. Why do we see 32.768kHz XTAL on some of the evaluation boards and embedded designs ? Which applications would need them ? Give examples. Which applications will not require the usage of this XTAL, give examples.

Ans: The 32.768 kHz crystal oscillator (XTAL) is commonly used in evaluation boards and embedded designs for various applications due to its specific frequency and characteristics. Here's why it's used and which applications would benefit from it, along with examples, followed by applications that may not require its usage:

Reasons for Usage:

1. Real-Time Clock (RTC):

- The 32.768 kHz crystal oscillator is commonly used as the clock source for real-time clock (RTC) circuits. This frequency is exactly 2^15 Hz, which is convenient for timekeeping applications, as it allows easy conversion of seconds to binary values.
- **Examples:** Embedded systems requiring accurate timekeeping, such as IoT devices, wearables, medical devices, and industrial control systems, often use the 32.768 kHz crystal oscillator for RTC functionality.

2. When RTC needs to have high precision like setting an event for particular clock time like 6:00AM Alarm, these kind of applications would need 32.768 kHz XTAL at RTC

Other reasons:

3. Low Power Consumption:

- The low frequency of 32.768 kHz results in lower power consumption compared to higher frequency oscillators. This makes it suitable for battery-powered devices where power efficiency is critical.
- **Examples:** Battery-powered devices like smartwatches, fitness trackers, and wireless sensor nodes benefit from using the 32.768 kHz crystal oscillator to minimize power consumption during sleep modes while maintaining accurate timekeeping.

4. Watchdog Timer:

- Some microcontrollers utilize the 32.768 kHz crystal oscillator as the clock source for the watchdog timer, which is used to reset the system in case of software or hardware failures.
- **Examples:** Safety-critical systems, automotive electronics, and industrial control systems often incorporate watchdog timers with the 32.768 kHz crystal oscillator to ensure system reliability and fault tolerance.

Applications that May Not Require 32.768 kHz XTAL:

1. When the time required is for delay and not bound to specific accuracy as per clock like inserting an approx delay, then precision XTAL is not needed, rather LSI or HSI sources are ok to be used

2. High-Speed Processing:
- Applications that require high-speed processing and data throughput, such as multimedia processing, graphics rendering, and high-performance computing, typically use higher frequency oscillators (e.g., MHz range) rather than the 32.768 kHz crystal oscillator.
- Examples: Desktop computers, gaming consoles, and servers do not typically require the use of the 32.768 kHz crystal oscillator for clocking purposes.

3. Communication Interfaces:
- Communication interfaces such as UART, SPI, I2C, Ethernet, and USB do not depend on the 32.768 kHz crystal oscillator for clocking. These interfaces often operate at higher frequencies determined by the system clock or external clock sources.
- Examples: Networking equipment, data acquisition systems, and communication modules do not require the 32.768 kHz crystal oscillator for communication purposes.

3. High-Frequency Timer Applications: - Applications that require high-frequency timers for precise timing and synchronization, such as audio/video processing, digital signal processing (DSP), and motor control, typically use higher frequency oscillators rather than the 32.768 kHz crystal oscillator.
- Examples: Audio/video equipment, robotics systems, and servo controllers use higher frequency oscillators for precise timing and synchronization tasks.

In summary, the 32.768 kHz crystal oscillator is commonly used in applications requiring accurate timekeeping, low power consumption, and reliable operation, such as real-time clock circuits, battery-powered devices, and safety-critical systems. However, it may not be necessary or suitable for applications requiring high-speed processing, communication interfaces, or high-frequency timing tasks.

Q9. What are the features of SPI or I2C communication. How to do describe the connections between 2 devices communicating on the interface ? Explain with drawing.

Ans: SPI (Serial Peripheral Interface) and I2C (Inter-Integrated Circuit) are serial communication protocols connecting multiple devices in embedded systems. Here are the features of each protocol and how connections are described between two devices communicating on the interface:

SPI (Serial Peripheral Interface):

Features:
1. Full-duplex synchronous communication.

2. Typically uses four wires: MOSI (Master Out Slave In), MISO (Master In Slave Out), SCLK (Serial Clock), and SS/CS (Slave Select/Chip Select).
3. Supports high-speed communication.
4. Allows multiple slave devices to be connected to a single master device.
5. Commonly used for short-distance communication between microcontrollers and peripheral devices such as sensors, displays, and memory chips.
6. Suitable for applications requiring high data throughput and low latency.

I2C (Inter-Integrated Circuit):

Features:
1. Half-duplex synchronous communication.
2. Typically uses two wires: SDA (Serial Data) and SCL (Serial Clock).
3. Supports multi-master communication, allowing multiple master devices to control the bus.
4. Slower than SPI but requires fewer wires, making it suitable for space-constrained applications.
5. Allows multiple slave devices connected to the same bus using unique 7-bit or 10-bit addresses.
6. Commonly used for interconnecting peripherals, sensors, and integrated circuits on a circuit board or between boards in a system.

Connection Description:

When describing connections between two devices communicating on SPI or I2C interfaces, the following aspects are typically included:

1. Master and Slave Devices:
- Identify which device acts as the master and which as the slave. The master device initiates communication and controls the timing of data transfer, while the slave device responds to commands from the master.

2. Signal Lines:
- For SPI: MOSI, MISO, SCLK, and SS/CS lines are specified, indicating the direction of data flow and the clock signal.
- For I2C: SDA and SCL lines are specified, representing the data and clock lines, respectively.

3. Pin Connections:
- Provide details on how the signal lines of each device are connected. For SPI, identify which pins are connected to MOSI, MISO, SCLK, and SS/CS on each device. For I2C, specify the connections for SDA and SCL lines.

4. Termination and Pull-up Resistors:
- Mention if any termination resistors or pull-up resistors are used on the communication lines to ensure signal integrity and noise immunity, especially in I2C connections.

5. Addressing (for I2C):
- In I2C connections, specify the unique address assigned to each slave device on the bus, allowing the master device to communicate with the desired slave.

Example Connection Diagram:

```
SPI Connection:

      Master Device                    Slave Device
  _____________________            _____________________
 |                     |          |                     |
 | MOSI --------->     |--------->| MOSI                |
 | MISO <---------     |<---------| MISO                |
 | SCLK --------->     |--------->| SCLK                |
 | SS/CS -------->     |--------->| SS/CS               |
 |_____________________|          |_____________________|


I2C Connection:

      Master Device                    Slave Device
  _____________________            _____________________
 |                     |          |                     |
 | SDA  ------------   |--------->| SDA                 |
 | SCL  ------------   |--------->| SCL                 |
 |_____________________|          |_____________________|
```

In the diagrams above, the connections between the master and slave devices for both SPI and I2C interfaces are illustrated. Each signal line is labelled accordingly, indicating the direction of data flow. Additionally, in the I2C connection diagram, the addressing aspect is mentioned, specifying the communication addresses of the slave devices.

Q10. What are the features of UART communication? How do you describe the connections between 2 devices communicating on UART? Explain withdrawing.

Ans: UART (Universal Asynchronous Receiver/Transmitter) communication is a widely used serial communication protocol in embedded systems. Here are some key features:

1. Asynchronous Communication: UART communication does not require a common clock signal between the transmitter and receiver. Instead, it relies on the agreed-upon baud rate for synchronization.

2. Simplex or Half-Duplex: UART supports simplex communication (one-way) or half-duplex communication (two-way, but not simultaneously).

3. Serial Data Transmission: Data is transmitted serially, one bit at a time, typically starting with a start bit, followed by 8 data bits (configurable), an optional parity bit, and finally one or more stop bits.

4. Configurable Baud Rate: The Baud rate defines the speed at which data is transferred over the UART interface. Common baud rates include 9600, 19200, 38400, 115200, etc.

5. Hardware Implementation: UART communication requires dedicated hardware (UART module) in both the transmitter and receiver devices.

Now, let's describe the connections between two devices communicating via UART with a drawing:

```
   Device A (Transmitter)           Device B (Receiver)

        TX  ------------------------> RX
 (Transmit Data)                (Receive Data)

        RX  <------------------------ TX
 (Receive Data)                (Transmit Data)

      GND  ------------------------ GND
   (Common Ground)             (Common Ground)
```

- TX (Transmit) / RX (Receive): These connections establish the serial communication link between the two devices. The TX pin of Device A is connected to the RX pin of Device B, and vice versa. Data transmitted by Device A's UART module is received by Device B's UART module, and vice versa.

- GND (Ground): This connection provides a common ground reference between the two devices, ensuring proper signal levels and reliable communication.

Remember, when connecting devices via UART, ensure that the baud rates, data formats (number of data bits, parity, stop bits), and voltage levels are properly configured and compatible between the transmitter and receiver to establish successful communication.

Q11. Draw the UART data packet communication diagram between the transmitter and receiver. Explain the bits of the communication packet. Explain withdrawing.

Ans: Sure, let's draw a UART data packet communication diagram and explain each part of the packet:

```
    Start Bit    Data Bits (8)    Parity Bit    Stop Bit(s)
      |            |                |             |
  ____|            |________________|_____________|____
 |  |              |                |             |  |
 |  |<- 1 bit ->   |<--- 8 bits --->|<- 1 bit ---->|  |
 |__|______________|________________|_____________|__|
    ↑                                  ↑
Transmitter                          Receiver
```

Now, let's explain each part of the UART data packet:

1. Start Bit: The Start Bit indicates the beginning of the data packet and prepares the receiver for incoming data. It is always a logic low (0) and alerts the receiver to start sampling data.

2. Data Bits (8): These are the actual data bits being transmitted. In this example, 8 data bits are shown, but the number of data bits can vary depending on the configuration (common options include 7 or 8 bits). Data is transmitted least significant bit (LSB) first.

3. Parity Bit: The Parity Bit, if used, is an optional bit for error checking. It can be set to odd, even, mark, space, or none. The parity bit is calculated based on the number of set bits in the data packet and is used for error detection during transmission.

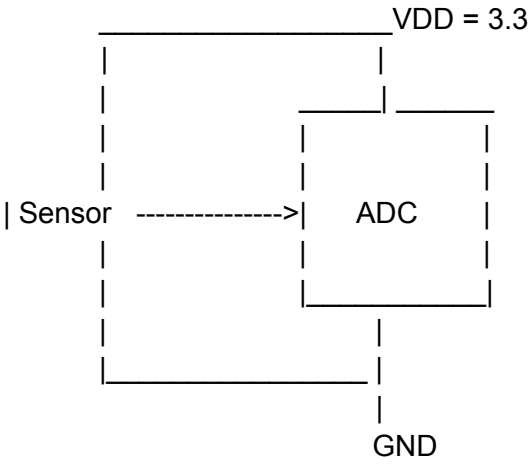
4. Stop Bit(s): The Stop Bit marks the end of the data packet. It is always a logic high (1) and gives the receiver time to prepare for the next start bit. Typically, one or two stop bits are used.

When data transmission begins, the UART line is typically held high (marking the idle state). When the Start Bit is detected, the receiver starts sampling the line at regular intervals determined by the baud rate. It samples each bit in the data packet and checks the parity bit (if used). Finally, it waits for the stop bit(s) to confirm the end of the packet.

This diagram illustrates the basic structure of a UART data packet. The implementation might vary based on settings such as baud rate, data format, parity, and stop-bit configuration.

Q12. When interfacing a sensor to 8bit ADC, Make a connection diagram. how will you calculate the weightage of 1bit of digital count converted by ADC which is referenced/powered by 3.3V ?

Connection Diagram:



Weightage of 1 bit:
Since the ADC is 8bit, the maximum conversion digital bits can be (2^8 - 1), so, the output can be 0xFF

Considering 3.3V as VDD, the minimum voltage that can be measured is 3.3V / 255 = 12.94 mV. This is also called as resolution of the ADC