Low-Level Device Interfaces & Prototyping

Mastering I²C, SPI, and UART for IoT Systems

This briefing provides a detailed, technical overview of the primary low-level serial communication protocols—I²C, SPI, and UART—critical for successful embedded systems and IoT prototyping. For engineers and advanced hobbyists, a thorough understanding of these interfaces is the bedrock of reliable hardware integration.

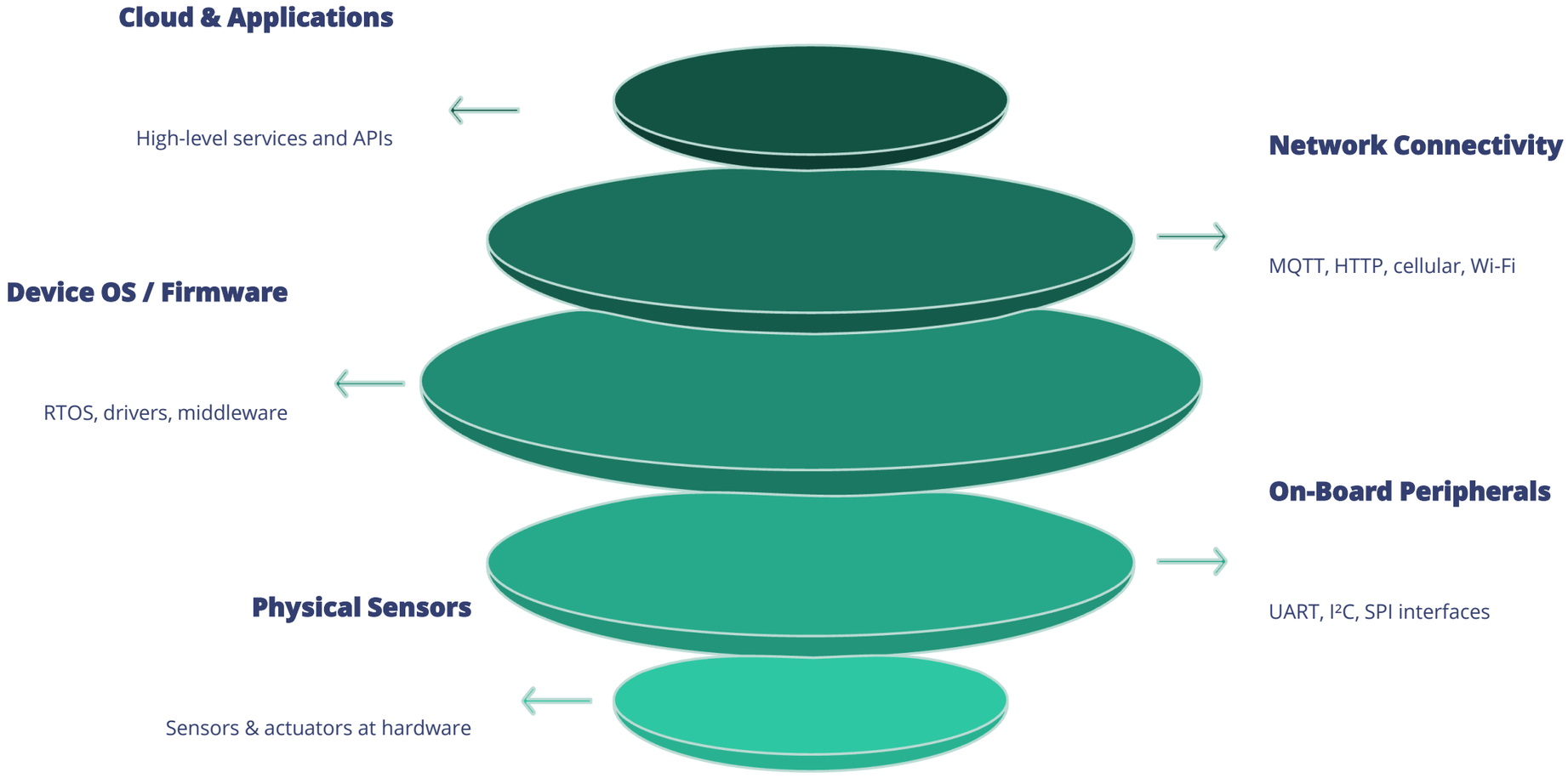
Learning Objectives

By the end of this lecture, you will be able to:

- Compare and contrast UART, I²C, and SPI across key characteristics like speed, pins, and topology.
- Interpret the electrical and timing diagrams for each protocol.
- Explain advanced concepts like I²C addressing/arbitration and SPI clock modes.
- Implement basic sensor communication using a microcontroller (Arduino/ESP32).
- Apply debugging techniques using logic analyzers and serial monitors to diagnose bus issues.
- Understand the role of ADCs and signal conditioning in sensor interfacing.

The IoT Connectivity Stack: A Systems View

The architecture of a modern IoT device involves several layers of abstraction. While high-level protocols like MQTT handle cloud communication, the fundamental connection between the microcontroller unit (MCU) and the physical world is managed by low-level peripheral interfaces.



> Key Insight: These low-level interfaces are the "nervous system" of your IoT device, connecting the MCU (the brain) to the sensors and actuators (the senses and limbs).

Mastering the protocols at the Device Peripherals layer is essential, as failures here manifest as data corruption, device non-response, and critical system instability. This is where hardware meets firmware, demanding precision in implementation.

Protocol Overview: Comparing the Big Three

Choosing the correct protocol for a given application depends on requirements for speed, complexity, and board space. The following table summarises the key differences between UART, I²C, and SPI.

Feature	UART (Universal Asynchronous Rx/Tx)	I²C (Inter-Integrated Circuit)	SPI (Serial Peripheral Interface)
Type	Asynchronous, Serial	Synchronous, Serial, Multi-drop	Synchronous, Serial, Point-to-point/Multi-Slave
Wires	TX, RX, GND (optional flow control)	SDA (Data), SCL (Clock)	MOSI, MISO, SCLK, SS/CS (per slave)
Topology	Peer-to-Peer	Multi-Master, Multi-Slave **Bus**	Single-Master, Multi-Slave **Star/Bus**
Speed	Low to Medium (up to ~5 Mbps)	Low to Medium (100 kbps - 3.4 Mbps)	High (up to 50+ Mbps)
Addressing	None (point-to-point)	**7/10-bit Slave Address**	Dedicated **Chip Select** (SS/CS) line

For quick communication over short distances, especially within a single PCB, SPI's speed and simplicity often make it the default choice, while I²C is favoured for sensor arrays due to its minimal pin count and addressing capability.

Deep Dive: UART (Universal Asynchronous Receiver/Transmitter)

UART is a simple, two-wire serial protocol fundamental to embedded debugging and peripheral communication (e.g., GPS, GSM modules). Its key characteristic is the absence of a dedicated clock line, making it 'asynchronous'.

Electrical & Timing Characteristics:

- **Asynchronous:** Both transmitting and receiving devices must operate at the same pre-configured **Baud Rate** (e.g., 9600, 115200 bps). Timing discrepancies are managed by sampling the line near the middle of each bit period.
- **Voltage Levels:** Typically uses TTL (Transistor-Transistor Logic) levels (3.3V or 5V). When connecting devices with different operating voltages, a Logic-Level Shifter is required to prevent component damage.

Framing Structure:

Every data byte is encapsulated within a frame, ensuring reliable delimitation:

Start Bit (Always '0')

Signals the start of the transmission frame, allowing the receiver to synchronise its clock.

Data Bits (5-9 bits)

The actual data payload, usually transmitted Least Significant Bit (LSB) first.

Parity Bit (Optional)

An optional bit used for basic error checking, ensuring the number of 1s is either even (even parity) or odd (odd parity).

Stop Bit(s) (Always '1')

Signals the end of the frame (1, 1.5, or 2 bits long). Ensures the line returns to the idle high state before the next start bit.

Deep Dive: I²C (Part 1) - Bus Architecture

The I²C protocol (Inter-Integrated Circuit, or TWI - Two-Wire Interface) is defined by its use of only two bidirectional lines, SDA (Serial Data) and SCL (Serial Clock), making it highly pin-efficient for connecting multiple peripherals.

Key Electrical Characteristics:

- **Synchronous:** The master device drives the SCL clock line, ensuring all devices operate in sync.
- **Open-Drain Nature:** Both SDA and SCL lines are driven by open-drain or open-collector drivers, meaning they can only pull the line LOW. They cannot actively pull the line HIGH. External pull-up resistors are mandatory to restore the line to VCC when all devices release it. This feature is vital for multi-master arbitration.

Transaction Data Frame:

1

Start Condition

SDA goes LOW while SCL is HIGH.

2

Address Frame

Master sends 7-bit slave address + Read/Write bit.

3

ACK/NACK

Slave acknowledges the address by pulling SDA low (ACK).

4

Data Frames

8 bits of data followed by an ACK/NACK bit.

5

Stop Condition

SDA goes HIGH while SCL is HIGH.

I²C's addressing scheme allows a single master to individually communicate with up to 112 unique devices on the same two-wire bus, provided they all operate at the same voltage level.

Deep Dive: I²C (Part 2) - Advanced Concepts

While I²C is relatively simple for single-master configurations, its strength lies in supporting complex multi-master environments and incorporating effective flow control mechanisms.

Arbitration and Flow Control

Bus Arbitration

If multiple masters begin transmitting simultaneously, the open-drain topology facilitates arbitration. Since a logic '0' (LOW) overrides a logic '1' (HIGH), the master that tries to transmit a '1' while the other transmits a '0' detects a mismatch and immediately ceases transmission. This ensures only one master controls the bus at any time, and the data transmission is non-destructive.

Clock Stretching

A crucial flow control mechanism. If a slave device needs more time to process the last received byte or prepare the next data byte (e.g., converting ADC readings), it can hold the SCL line LOW. The master must wait until the slave releases the SCL line, effectively pausing the transaction until the slave is ready.

Common Error Diagnosis

- **NACK (No Acknowledgement):** The most frequent issue, indicating the slave did not respond after the address or a data byte. Causes include incorrect address, device disconnection, or incorrect pull-up resistor values.
- **Bus Stuck:** SDA or SCL line is perpetually held LOW. This often happens if a slave fails mid-transaction or if power is unstable. This usually requires a hard reset or a specific software sequence to attempt to manually clear the bus state.

Deep Dive: SPI (Serial Peripheral Interface)

SPI is the protocol of choice for applications requiring high data rates, such as memory chips, displays, and high-speed ADCs. Its full-duplex nature and dedicated data lines contribute to its exceptional speed.

Electrical & Data Characteristics:

- **Synchronous & Full-Duplex:** The Master controls the SCLK and can send (MOSI) and receive (MISO) data simultaneously. This concurrent data flow defines its superior bandwidth.
- **Hardware Addressing:** Unlike I²C, SPI uses a dedicated physical line—the **Slave Select (SS) or Chip Select (CS)**—to select the target device. The master pulls one CS line LOW to activate the corresponding slave; all other slaves ignore traffic.

The Four Critical Pins:

MOSI

Master Out, Slave In. Data flows from the Master to the currently selected Slave.

MISO

Master In, Slave Out. Data flows from the selected Slave back to the Master.

SCLK

Serial Clock. Generated by the Master to synchronise data transfer.

SS/CS

Slave Select / Chip Select. Active LOW signal from Master to select a specific Slave.

Because each slave requires a dedicated CS line, the number of slaves is limited by the number of available GPIO pins on the master MCU.

SPI Clock Modes (CPOL & CPHA)

The SPI protocol requires both the Master and Slave to agree on not just the speed, but also the clock mode, which dictates the polarity and phase of the clock signal. These are defined by two parameters: CPOL and CPHA, resulting in four distinct modes.

CPOL (Clock Polarity): Idle State

- **CPOL=0:** The clock signal is LOW when idle.
- **CPOL=1:** The clock signal is HIGH when idle.

CPHA (Clock Phase): Data Sampling Edge

- **CPHA=0:** Data is sampled on the leading (first) clock edge (e.g., the rising edge if CPOL=0).
- **CPHA=1:** Data is sampled on the trailing (second) clock edge (e.g., the falling edge if CPOL=0).

Key Takeaway: Both the master and slave **must** be configured for the same **Mode (0, 1, 2, or 3)**. Mode 0 (CPOL=0, CPHA=0) is the most commonly used default configuration across peripheral manufacturers. If communication fails, checking the clock mode is the first troubleshooting step.

Debugging Tools: Your Best Friends

In embedded development, the phrase "it compiled, but it doesn't work" is common. Low-level bus errors are often invisible without the right tools. Effective debugging relies on specialised instruments that provide insight into the electrical signals.

1

1. Logic Analyzer (Digital Protocols)

Essential for I²C and SPI. A logic analyzer captures digital waveforms and—critically—uses software-based **Protocol Decoders** to translate the raw high/low signals into readable data packets, including start/stop conditions, addresses, and data bytes. This tool verifies timing, confirms ACKs, and isolates bus errors quickly.

2

2. Serial Monitor (UART/Console)

The most basic and widely accessible tool. Used primarily for debugging code execution flow via `Serial.print()` statements and for communicating directly with console-based modules (e.g., AT commands to a GSM modem). It only verifies the content, not the electrical timing.

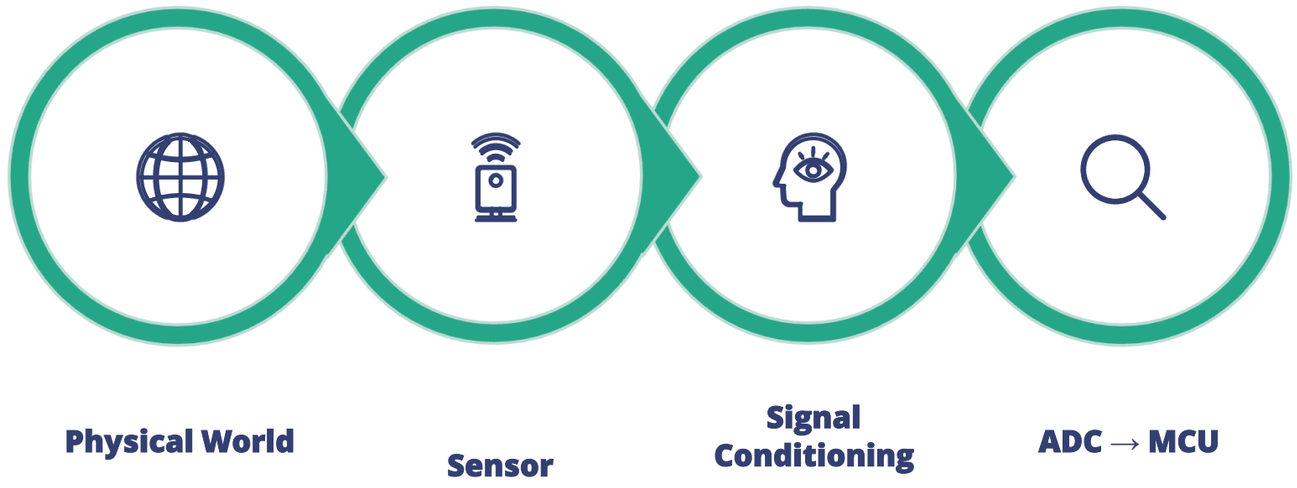
3

3. Digital Oscilloscope (Analog/Signal Integrity)

While a logic analyzer verifies digital state, the oscilloscope verifies signal quality. Use it to analyse analog characteristics: noise floor, voltage spikes, rise/fall times, and transient power issues. Essential for ensuring signal integrity in high-speed SPI implementations.

Sensor Fundamentals: ADC & Signal Conditioning

Many physical world measurements (temperature, light, pressure) are analog. Interfacing these sensors requires conversion and preparation steps before the data can be processed by the MCU.



Analog-to-Digital Converter (ADC)

The ADC converts a continuous analog voltage into a discrete digital number. The quality and granularity of this conversion are determined by several key specifications:

12-bit

Resolution

Determines the number of discrete steps the voltage range is divided into (e.g., 2^12 = 4096 steps). Higher resolution means better accuracy.

1MSPS

Sampling Rate

How quickly the ADC can perform conversions (Samples Per Second). Must adhere to the Nyquist theorem for accurate signal capture.

3.3V

Reference Voltage

The maximum voltage the ADC can measure. All input voltages are measured relative to this reference.

Signal Conditioning

Signal conditioning is often necessary to scale, clean, or protect the sensor output before it reaches the ADC input:

- **Amplification:** Using op-amps to boost very weak signals (e.g., from thermocouples or strain gauges) to fully utilise the ADC's voltage range.
- **Filtering:** Employing passive (RC) or active filters to remove high-frequency noise that could alias or corrupt the measurement.
- **Level Shifting/Protection:** Ensuring the signal is safely within the ADC's operating voltage range and protecting against over-voltage spikes.