AUTOMATED SYSTEMS

Computer Practicum EN - WIS - LES

Lecturer: Niki Hrovatin

What Is an Automated System?

An automated system is a system that performs a task or controls a process **according to programmed rules**, with little need for continuous human intervention.

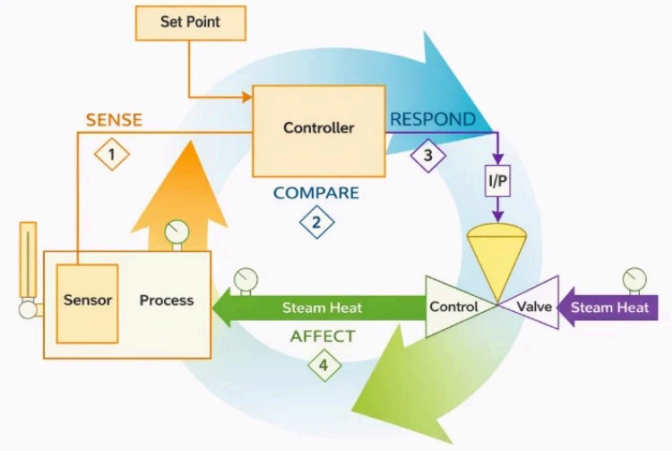
- Once started, it can carry out actions without a person directing every step.
- It uses **inputs**, some form of **decision-making**, and **outputs**.
- Automation can be used to monitor a situation, perform a sequence of actions, or regulate a physical process.

Examples: automatic lighting, washing machines, and temperature-controlled ventilation.

From Computers to the Physical World

Computers process **data**. But how can software interact with the **physical environment**?

- It needs a way to **observe** what is happening.
  - collect information using **sensors**.
- It needs a way to **decide** what should happen next.
  - process the information and apply rules using a **controller**.
- It may need a way to **change** the environment.
  - produce a physical effect using **actuators**.

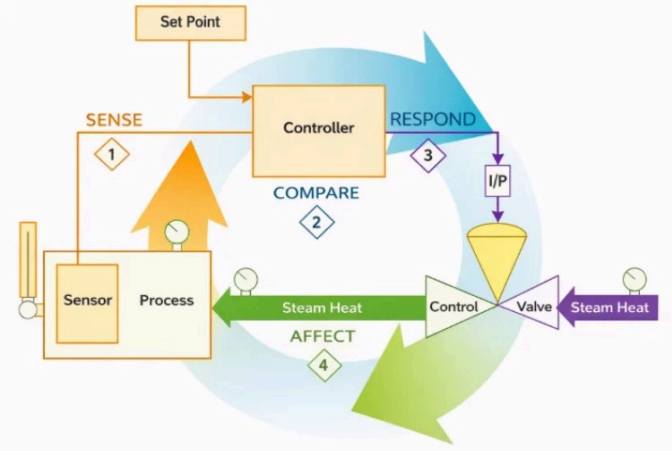


Example: a thermostat repeatedly adjusts heating based on the measured room temperature.

The Feedback Control Loop

A **feedback control loop** uses measurements of the system to adjust its future actions.

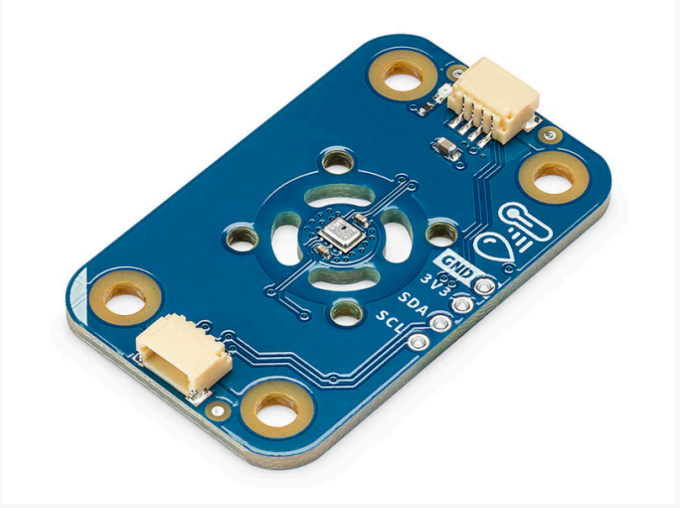
1. A **sensor** measures the current condition.
2. The **controller** compares it with a desired condition (the **setpoint**).
3. An **actuator** changes the physical process.
4. The system is measured again to check the result.



Example: a thermostat repeatedly adjusts heating based on the measured room temperature.

Sensors: Measuring the Environment

A **sensor** is a device that **detects a physical quantity** and converts it into a **signal** that can be interpreted from a control device.



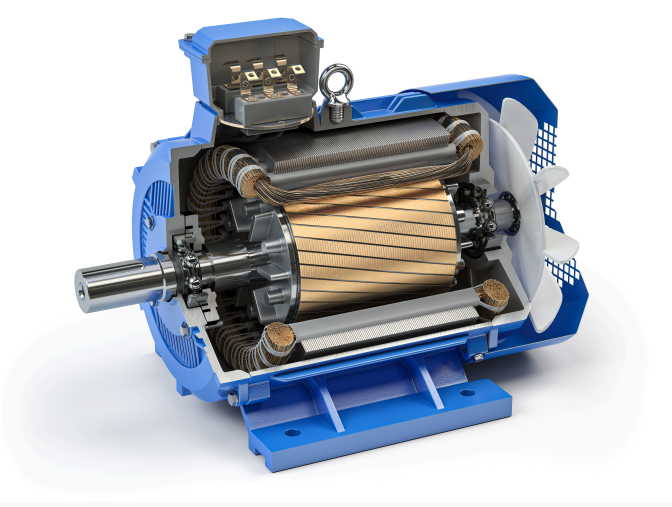
Examples of quantities that sensors measure:

- **Temperature** and **humidity**
- **Light** intensity or color
- **Pressure**, **force**, or **motion**

A sensor provides **observations**. The controller interprets them and decides what to do.

Actuators: Acting on the Environment

An **actuator** is a device that converts a control signal into a **physical effect** on its surroundings.



Examples:

- A **motor** moves a fan or mechanism.
- A **valve** changes the flow of air or water.
- A **LED** (Light-Emitting Diode) produces light to illuminate an area or indicate a system's status.
- A **relay** switches an electrical load.

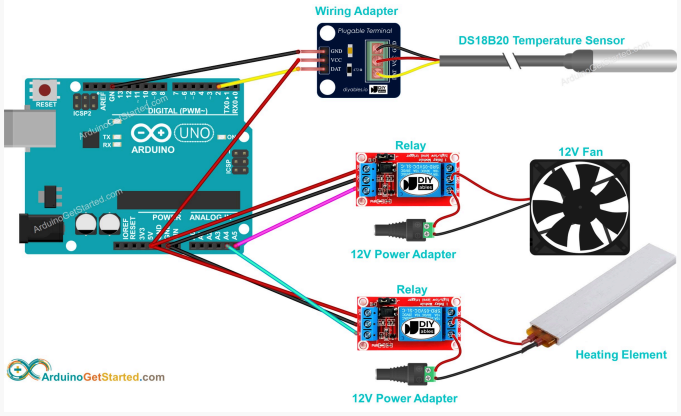


Sensors observe the environment; actuators change it.

Who Makes the Decisions? The Controller

The **controller** is the part of the system that:

- **Receives inputs** (such as sensor measurements).
- **Executes programmed rules** or a control algorithm.
- **Produces outputs** (such as commands to actuators).



Example: if temperature exceeds **26 °C**, command the fan to switch on.

A controller is a **function** in a system: it can be implemented using a microcontroller, a computer, or other control hardware.

What Is a Microcontroller (MCU)?

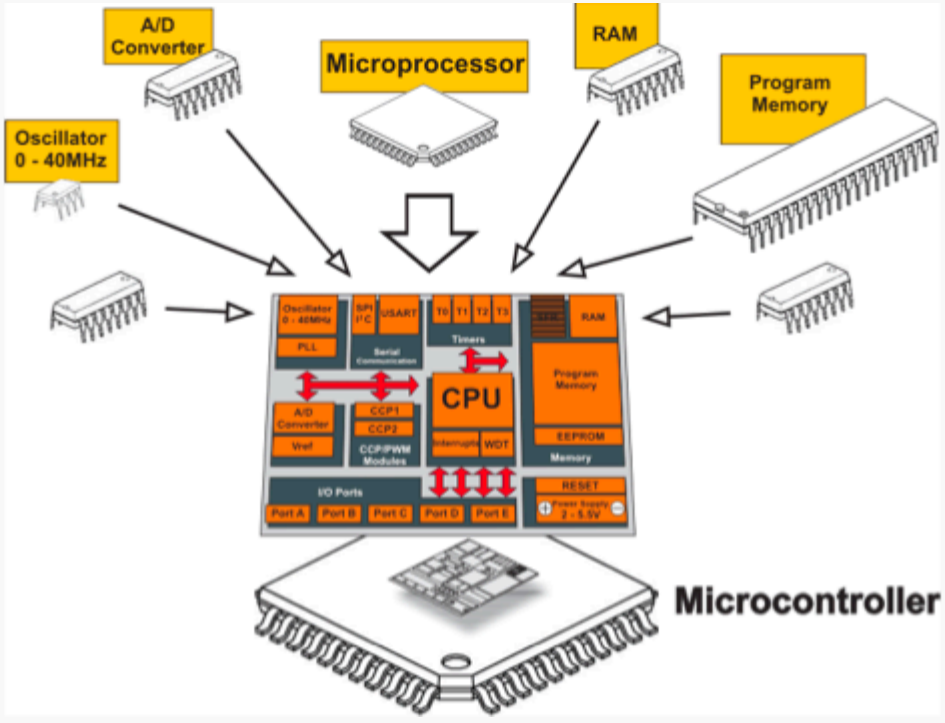
A **microcontroller (MCU)** is a **small computer integrated into a single chip**, designed to run programs that monitor or control electronic devices.

A typical microcontroller includes:

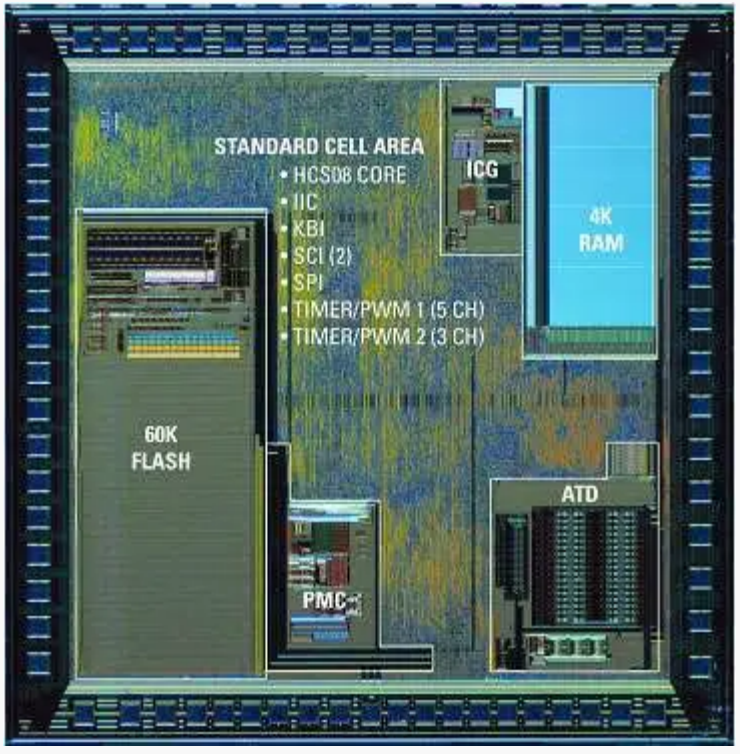
- **CPU (processor):** Executes program instructions.
- **Flash memory:** Stores the program (*firmware*) / data/ state even when powered off.
- **RAM:** Volatile memory where the firmware is running.
- **Clock (oscillator):** Provides the timing signal that synchronizes operations.
- **Input/Output (I/O) peripherals:** Enable interaction with external components, including:
  - **GPIO:** Digital inputs and outputs.
  - **ADC:** Converts analog signals into digital values.
  - **Communication interfaces:** Support protocols such as I^2C, SPI, and UART.
  - **Timers and PWM:** Provide timing and control functions.

A microcontroller combines processing, memory, and hardware interfaces in a single chip.

Components of a microcontroller



Zoom Inside a Microcontroller



HCS08QG8 8-bit Microcontroller Student Learning Kit





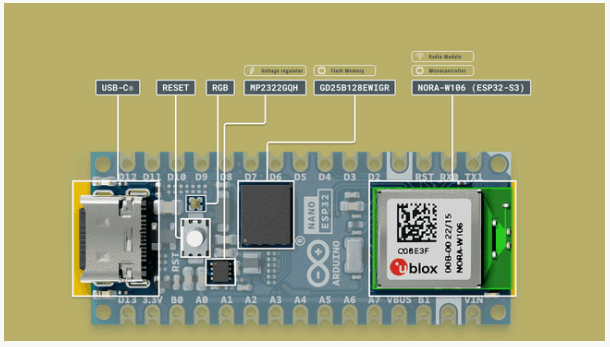
Microcontroller Chip vs. Development Board

A **microcontroller** is the chip that runs the program.
A **development board** makes that chip easy to connect, power and program.

Our Arduino Nano ESP32 board:

- Uses an **ESP32-S3 based microcontroller**.
- Adds accessible pins, a USB connection and supporting circuitry.
- Lets us connect external sensors and write programs using **MicroPython**.

Arduino nano ESP32 (devkit)



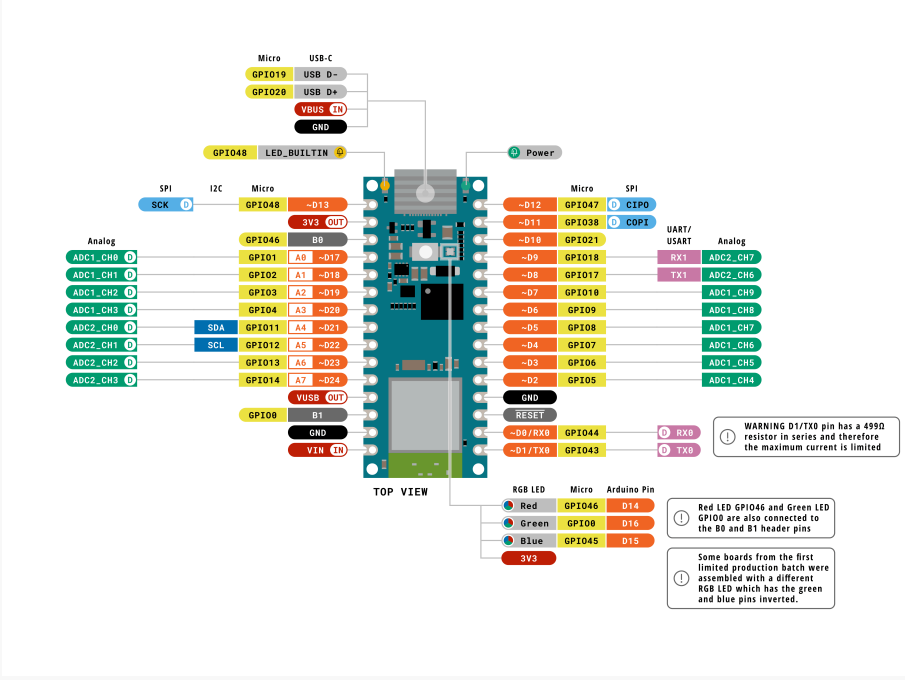
Nora-w106 (microcontroller)



Peripherals: Connecting to Other Devices

Microcontrollers include **peripherals**: hardware that perform specific input/output tasks.

- **GPIO:** read or set digital signals (buttons, LEDs).
- **ADC:** convert an analog voltage into a digital value.
- **Communication interfaces:** exchange data with other devices (**I²C**, SPI, UART).
- **Timers / PWM:** measure time or generate timed output signals.



These allow a program to interact with the physical world.

Actuators: From Electrical Commands to Physical Effects

While **sensors observe** the environment, **actuators change** it.

An actuator uses **supplied energy** to produce a physical effect in response to a **control signal**.

Examples of electrical actuators:

- **Motor:** electrical energy → **motion** (e.g., a fan).
- **LED:** electrical energy → **light** (e.g., a status indicator).
- **Heating element:** electrical energy → **heat** (e.g., a heater).
- **Solenoid:** electrical energy → **mechanical movement** (e.g., opening a valve).

The control signal determines the action; the energy supply makes it possible.

ceramic heat plate

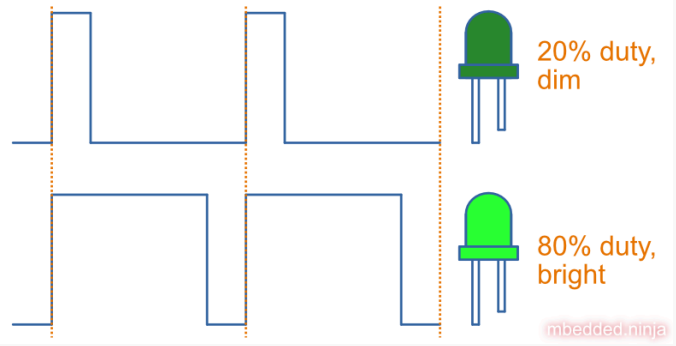


solenoid valve



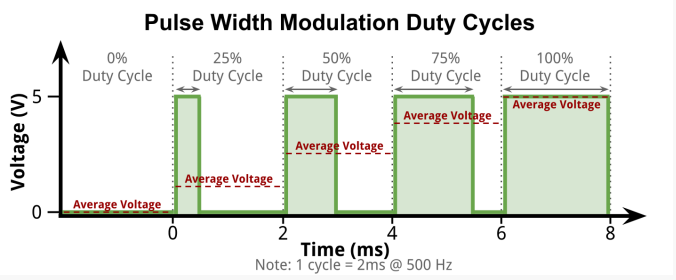
How Does a Microcontroller Control an Actuator?

A microcontroller sends **control signals**, but its pins usually cannot supply enough power for motors and other high-power actuators.



Common ways to control actuators:

- **GPIO (ON/OFF):** switches an output between two states.
- **PWM (Pulse-Width Modulation):** changes the proportion of time a signal stays ON; useful for controlling LED brightness or motor speed.
- **Driver circuit:** uses the control signal to switch or regulate power from a suitable supply (e.g., a transistor or motor driver).



How Do Sensors Measure the Physical World?

Sensors rely on **transducers** to detect physical quantities.

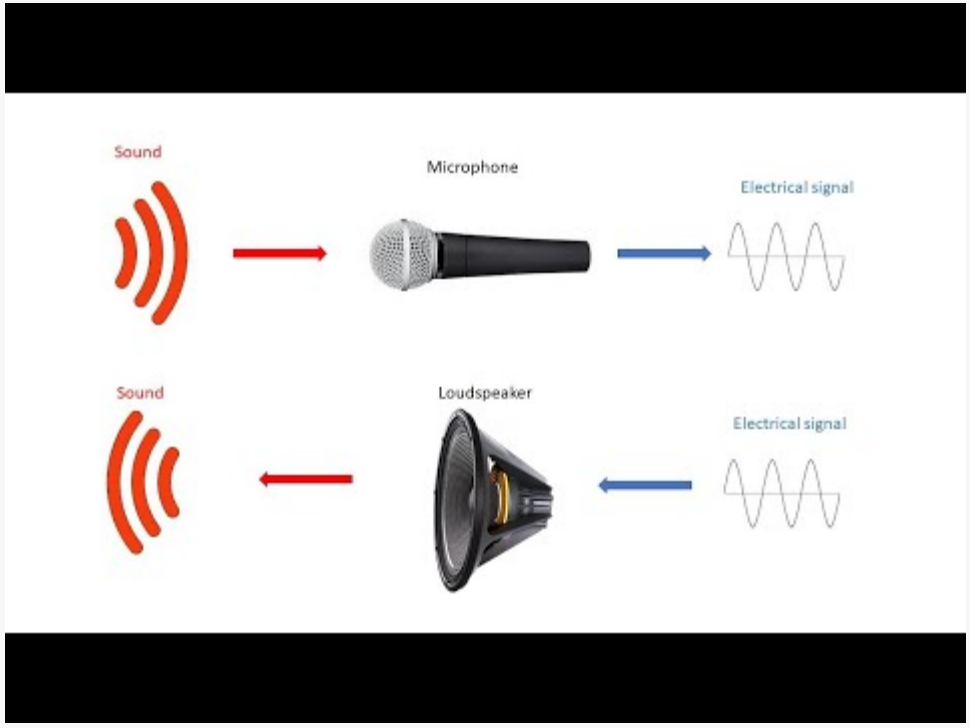
A **transducer** is a component that converts a physical quantity or one form of energy into another.

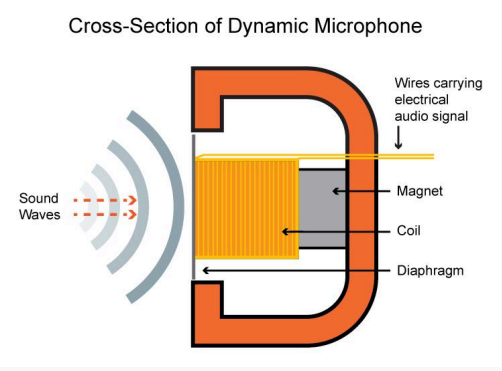
In electronic sensors, transducers convert physical quantities into measurable electrical properties, such as:

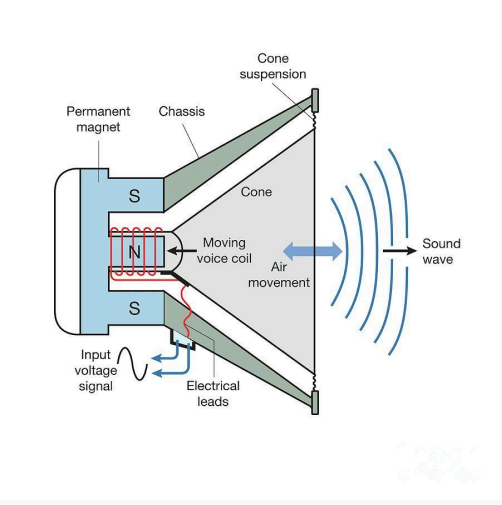
- **Resistance** (Ω)
- **Voltage** (V)
- **Current** (A)
- **Capacitance** (F)

These electrical changes are measured and interpreted to determine the physical quantity, expressed in an appropriate unit.

How Do Sensors Measure the Physical World?





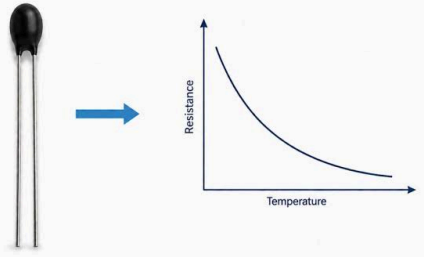


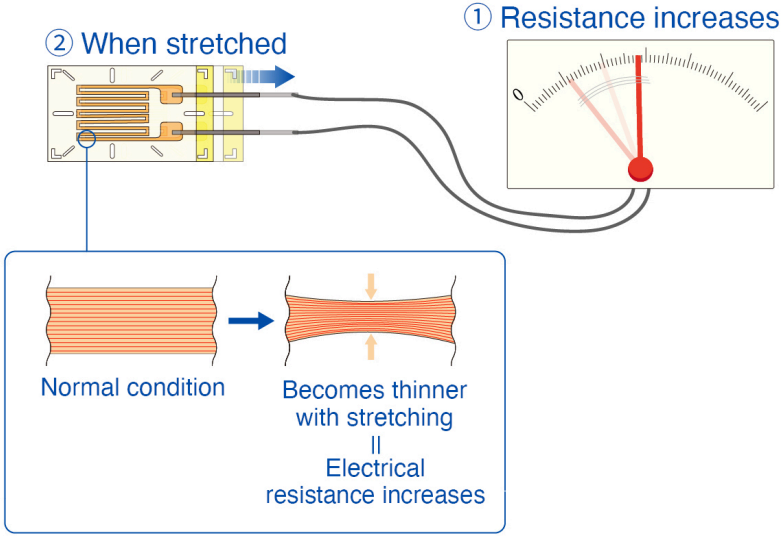
A dynamic loudspeaker can work as a microphone, because both devices use essentially the same physical principle: electromagnetic transduction.

Examples of Sensing Transducers

Different sensing elements rely on different physical effects:

- **NTC thermistor — temperature → resistance.** Its electrical resistance **decreases as temperature rises**.
- **Strain gauge — deformation → resistance.** Stretching or compressing a material produces a **small resistance change**.





Their raw outputs are not necessarily temperatures in °C or forces in newtons.

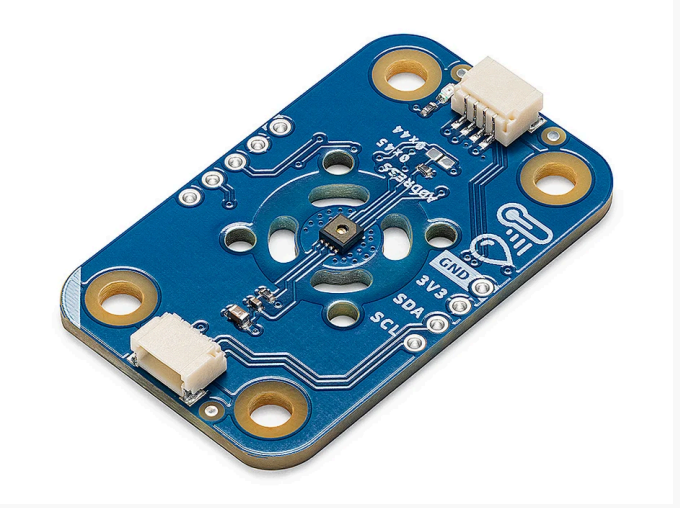
- **Additional interpretation is needed to obtain useful measurements.**

From a Transducer to a Sensor

A **sensor** is a device used to detect or measure a physical quantity and provide an **interpretable output**.

A complete **sensor module** may combine:

1. **Sensing element (transducer):** responds to temperature, light, force, etc.
2. **Signal conditioning:** adapts the electrical response (amplifies, filters, converts resistance to voltage, etc.).
3. **Conversion and processing:** may digitize, calibrate or linearize the measurement.
4. **Output interface:** provides an analog signal or digital data to a controller.



The sensing element (transducer) detects the effect; the sensor system makes the measurement usable.

Why Does a Sensor Need Signal Conditioning?

The electrical response of a sensing element is not always ready to be read by a microcontroller.

Signal conditioning prepares this response for measurement:

- **Convert:** a thermistor changes resistance, but a controller may need a different type of signal
- **Amplify:** a strain gauge or photodiode may produce a very weak signal.
- **Filter:** reduce unwanted noise or interference before measuring.
- **Calibrate / linearize:** compensate for a nonlinear response or systematic differences between devices.

These functions may be implemented in a sensor module, an external circuit, or partly in software.

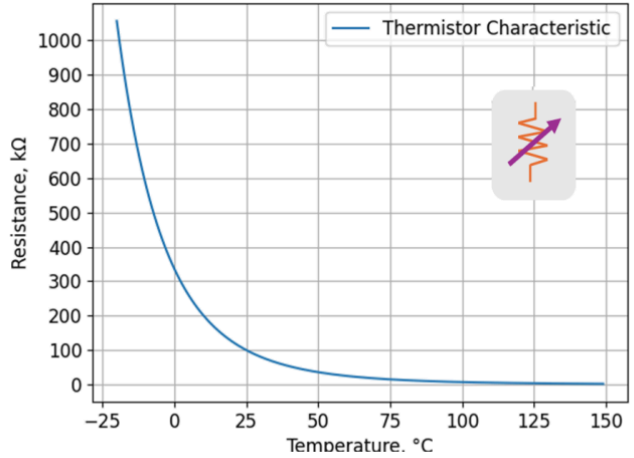
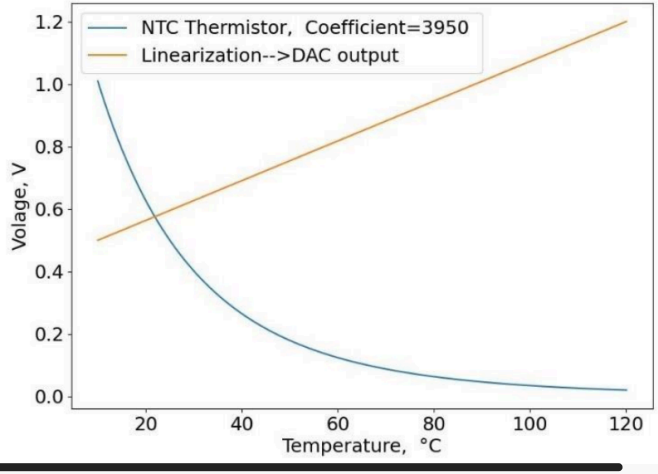


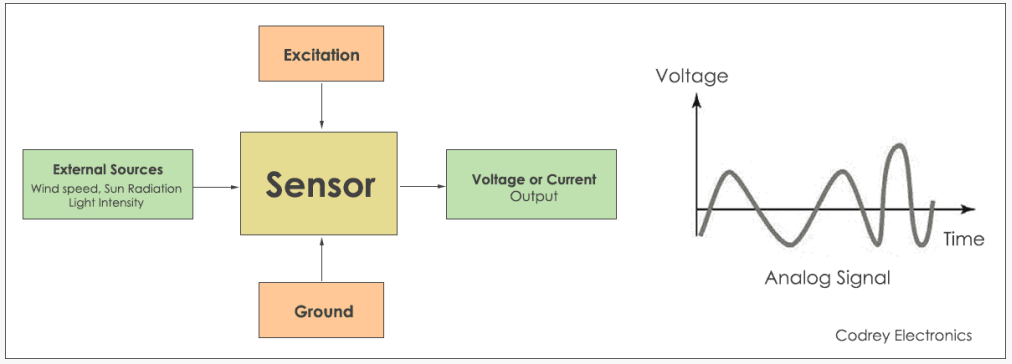
Figure 4. NTC thermistor characteristic. (R_{25C} = 100 KΩ; β = 3950)



Analog Sensors vs. Digital Sensors

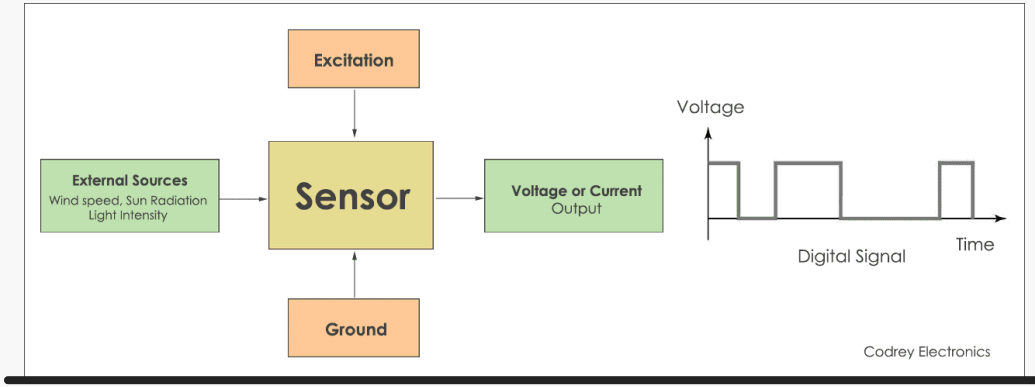
Analog-output sensor: Outputs a continuous signal (voltage or current).

- Read using Analog-To-Digital-Converter **ADC**



Digital-output sensor: Outputs values encoded as digital signals

- Common protocols I^2C, SPI or UART.
- Usually performs signal conversion and some processing **inside the sensor**.



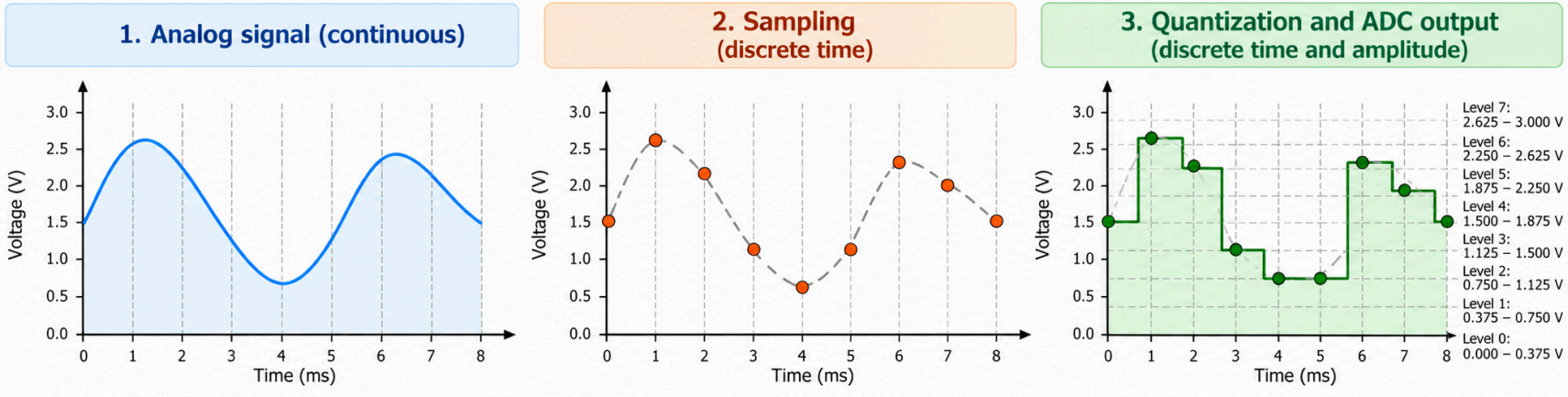
How to Read an Analog Sensor?

An **analog-to-digital converter (ADC)** transforms a varying electrical signal into a numerical representation.

The conversion involves two key operations:

1. **Sampling:** observe the signal at **specific instants of time**.
2. **Quantization:** represent each observed value using **one of a finite number of levels**.

Sampling discretizes time; quantization discretizes amplitude.



Sampling time t (ms)	0	1	2	3	4	5	6	7	8
ADC code (decimal)	4	7	6	3	2	2	6	5	4
ADC code (binary, 3-bit)	100	111	110	011	010	010	110	101	100
Quantized voltage (V)	1.500	2.625	2.250	1.125	0.750	0.750	2.250	1.875	1.500

How to Read an Analog Sensor?

The ADC produces a **binary value** representing the measured electrical signal.

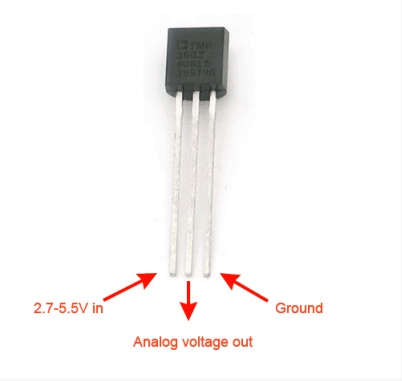
To obtain a meaningful physical measurement, the microcontroller must:

1. **Convert the ADC value into a voltage (V)** using the ADC's reference voltage and resolution.
2. **Interpret the electrical signal** using the sensor's **characteristic curve** or calibration equation.
3. **Determine the physical quantity**, expressed in its corresponding unit (°C, Pa, lux, etc.).

Example — Temperature measurement using a TMP35:

- ADC binary value → Voltage (V) → Temperature (°C)

The sensor's characteristic curve describes the relationship between its electrical response and the physical quantity being measured.



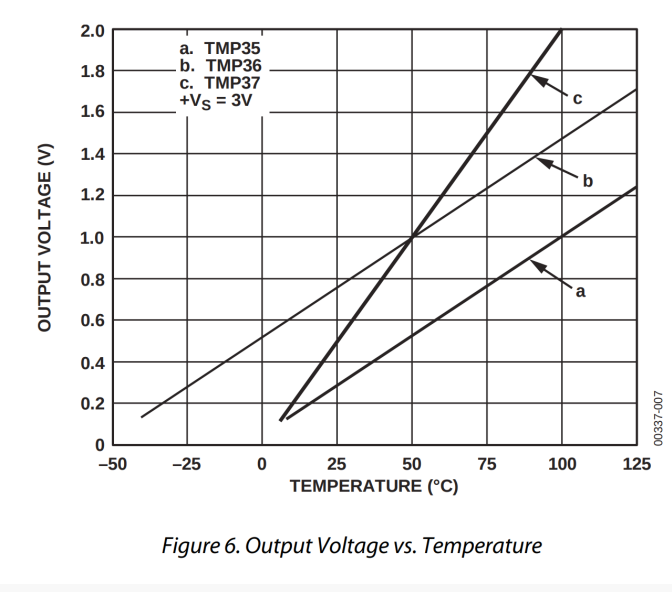


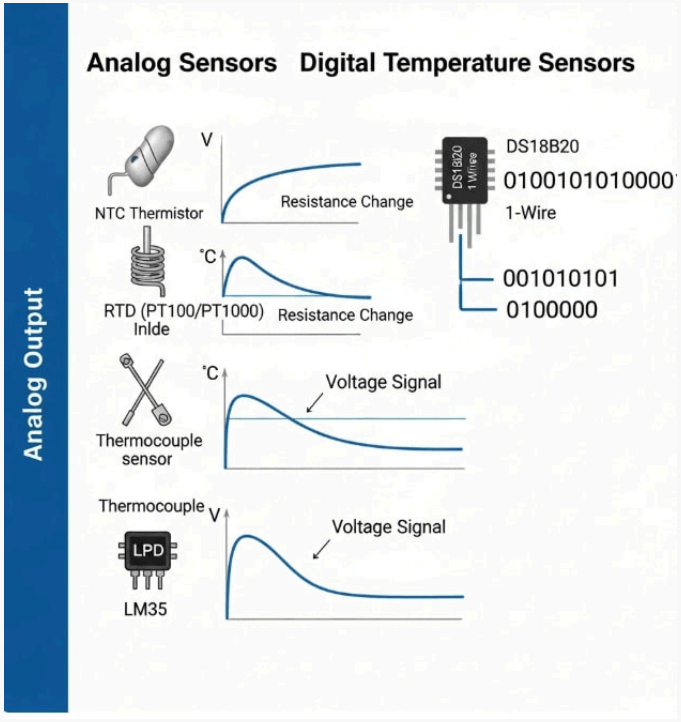
Figure 6. Output Voltage vs. Temperature

Digital Sensors

A **digital sensor** measures a physical quantity and provides the result as **digital data**, rather than as an analog electrical signal.

A digital sensor typically performs the following operations internally:

1. **Sensing:** Converts a physical quantity into an electrical signal.
2. **Analog-to-digital conversion (ADC):** Converts the electrical signal into a digital value.
3. **Signal processing:** Applies calibration or other processing to obtain a meaningful measurement.
4. **Data encoding:** Represents the measurement as digital data (bits and bytes), which can be interpreted in a physical unit (°C, Pa, lux, etc.).



Sampling Inside a Digital Sensor

A **digital sensor** often performs its own measurements using internal electronics, then makes the results available in digital form in a specific unit of measure.

Three rates must not be confused:

- **Sensor measurement rate:** how often the sensor produces a **new physical measurement**.
- **Controller reading rate:** how often the microcontroller requests the latest result.
- **Communication bit rate:** how fast data bits are transferred.

Example: if a sensor updates **10 times/s**, reading it **100 times/s** will usually return repeated values between updates.

Reading data faster does **not** make the sensor measure faster.

Digital Sensors: How Are Measurements Transferred?

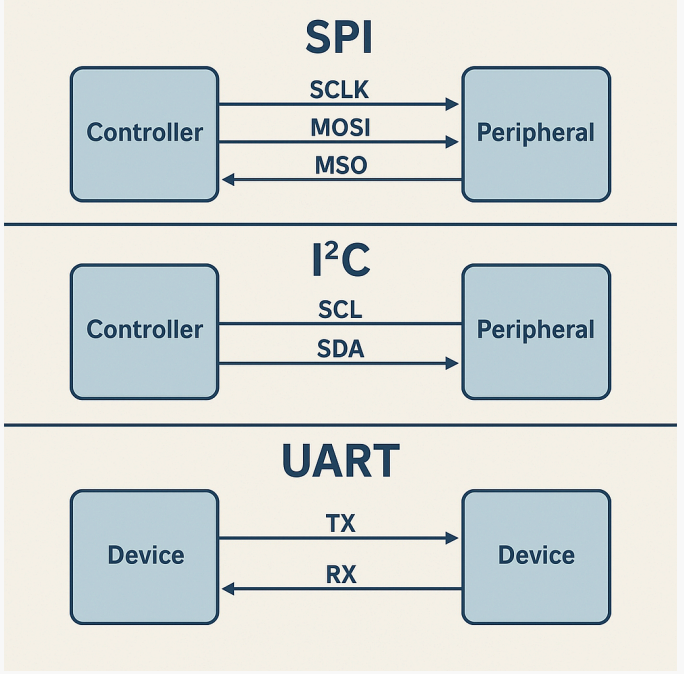
Once a sensor has produced a digital measurement, the microcontroller must be able to **receive the corresponding bits**.

A **communication protocol** defines the rules for exchanging data, including:

- **Connections:** which pins or wires carry the signals.
- **Timing:** when each bit is transmitted or sampled.
- **Device selection:** how to reach the intended sensor.
- **Data organization:** how commands and measurement bytes are exchanged.

Three common short-range wired interfaces used with microcontrollers are **UART, I²C and SPI**.

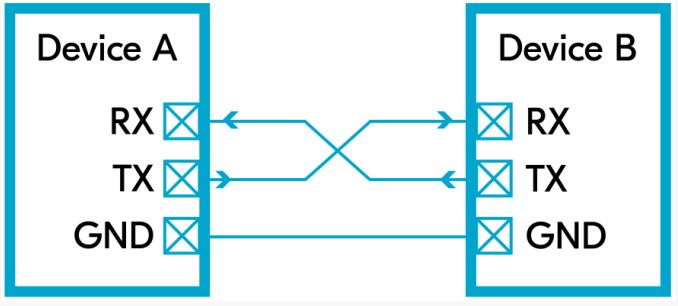
They all transfer digital data, but organize the communication differently.



UART: Direct Communication Between Two Devices

UART (**Universal Asynchronous Receiver/Transmitter**) sends data serially between two devices, usually over a **point-to-point** connection.

- **TX (transmit)** of one device connects to **RX (receive)** of the other, and vice versa; the devices also share a ground reference.
- It is **asynchronous**: no shared clock line; both devices agree on a **baud rate**.
- Each character is framed by **start and stop bits** (and optional parity).
- Sending and receiving can happen simultaneously (**full-duplex**).

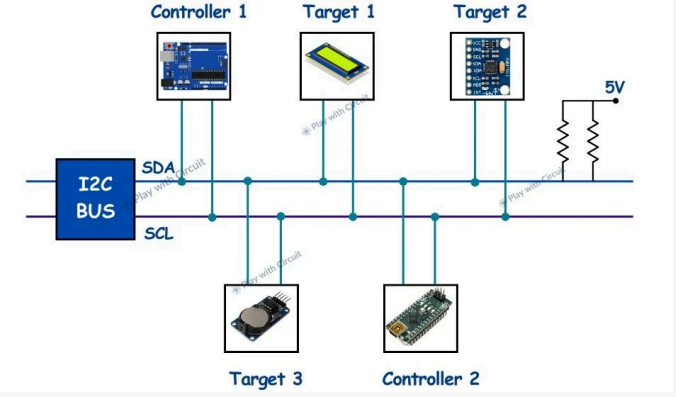


Common use: serial communication between a microcontroller and a GPS module or serial interface.

I²C: Multiple Devices on Two Shared Wires

I²C (**Inter-Integrated Circuit**) is a **synchronous** serial bus intended for communication between nearby electronic devices.

- All devices share **SDA** (data) and **SCL** (clock), plus a common ground.
- A **controller** initiates transfers and supplies the clock; the addressed **target** responds.
- Devices are selected using a **bus address** (commonly 7 bits).
- Data travels in **both directions**, but not simultaneously on SDA (**half-duplex**).
- The bus uses **pull-up resistors** and open-drain outputs.

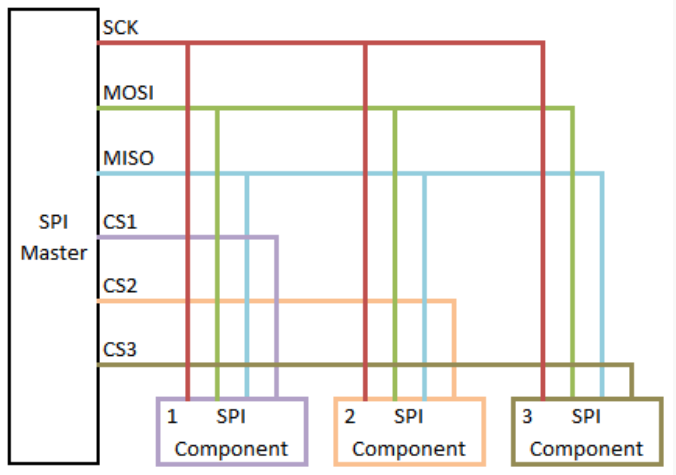


Typical use: connecting several low-data-rate sensors to one microcontroller.

SPI: Fast Communication with Selected Devices

SPI (**Serial Peripheral Interface**) is a **synchronous** serial interface commonly used when higher data-transfer rates are needed.

- A controller provides a clock on SCLK.
- MOSI: sends data from the controller to the peripheral.
- MISO: sends data from the peripheral to the controller.
- A CS (chip-select) line activates the intended peripheral.
- Data can move in both directions simultaneously (full-duplex).
- With several peripherals, the controller typically needs a separate CS line for each.



Typical uses: external Flash memory, displays, high-speed converters and SD cards (in SPI mode).

UART, I²C and SPI: How They Communicate

Feature	UART	I²C	SPI
Timing	Asynchronous	Synchronous (clock)	Synchronous (clock)
Num of cables	2	2	3 (+1 CS for each device)
Topology	Point-to-point	Shared bus, multiple targets	Controller + selected peripherals
Device selection	No built-in address	Target address	Separate CS line
Communication roles	Two peers	Controller / target	Controller / peripheral
Simultaneous TX and RX	Yes (full-duplex)	No (half-duplex)	Yes (full-duplex)

All require compatible voltage levels and a common electrical reference (or suitable interface hardware).

UART, I^2C and SPI: Speed, Distance and Use

Feature	UART	I^2C	SPI
Common speeds	9,600–115,200 bit/s; possible 5+ Mbit/s	100 or 400 kbit/s; optional 1 or 3.4 Mbit/s	Often 1–20+ Mbit/s; device-dependent
Typical wiring	Short logic-level cable	PCB or short cable	PCB or very short cable
Cable length	Typically up to a few meters	Usually less than 1 m	Usually a few tens of cm
Adding devices	Usually requires another link	Share SDA/SCL; use addresses	Usually add a CS signal
Pros	Simple, only 2 signal wires	Few wires for many sensors	High speed, low overhead
Cons	Typically connects only 2 devices	Relatively low speed	More wiring and higher complexity
Typical use	GPS / serial module	Temperature / light / air-quality sensors	Flash / display / fast ADC

Choosing the Interface for a Sensor

The best interface depends on **what must be connected** and **how quickly measurements must be exchanged**.

- **Several sensors on one board:** I^2C allows them to share two signal lines while being selected by address.
- **Large data transfers:** SPI is often preferable for fast converters, memory or displays.
- **Two devices exchanging serial data:** UART offers a simple point-to-point connection.

For the **Arduino Nano ESP32** laboratory, we will mainly use **I^2C** because many of our sensor modules already support it.

The next step is understanding how the controller reads sensor data using addresses, commands and registers.

I^2C: Why Was It Developed?

I^2C (Inter-Integrated Circuit) was developed by **Philips in 1982** to connect chips using very few wires.

Pronounced "eye-squared-see" or "eye-two-see" Sometimes also called "IIC" or two-wire-interface (TWI)

It is especially useful when:

- Devices are **close together** (often on the same board).
- Many devices must share a connection.
- **Low wiring complexity** matters more than maximum speed.

Today: widely used for sensors, displays and control electronics.

I²C: Talking Over Two Wires

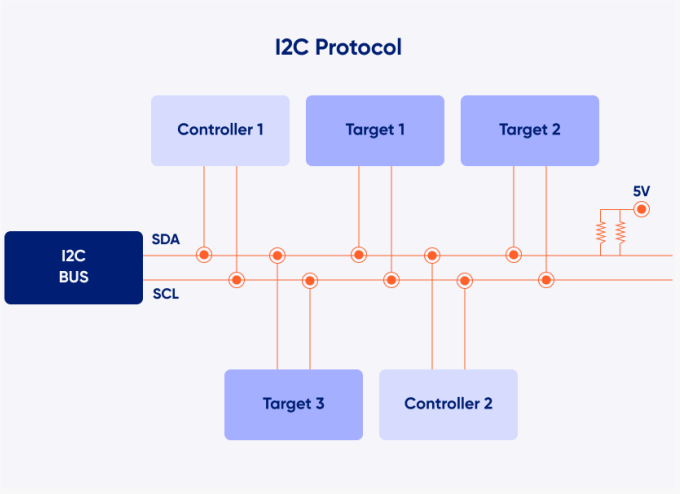
SCL (Serial Clock)

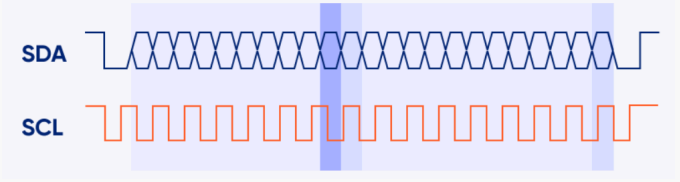
- Provides timing for data transmission.
- Normally generated by the **controller**.

SDA (Serial Data)

- Carries data in **both directions**.
- Is shared by all connected devices.

One controller can communicate with multiple targets over the same two signal lines.



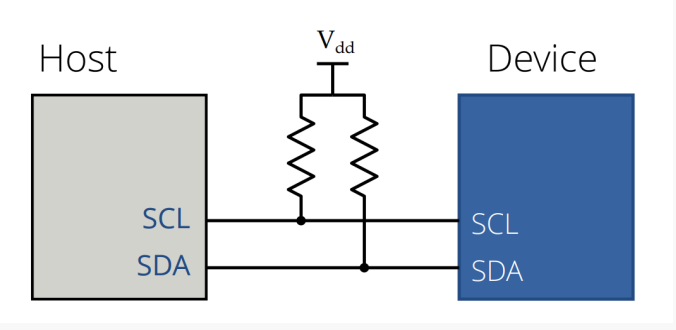


How Do We Share the SDA Line?

Both the controller and the sensor sometimes need to transmit over **the same SDA wire**.

To avoid opposing electrical outputs, I^2C uses a simple rule:

- A device may **pull SDA LOW (0)**.
- Otherwise, it must **release the line**.
- A **pull-up resistor** returns the line HIGH (1).



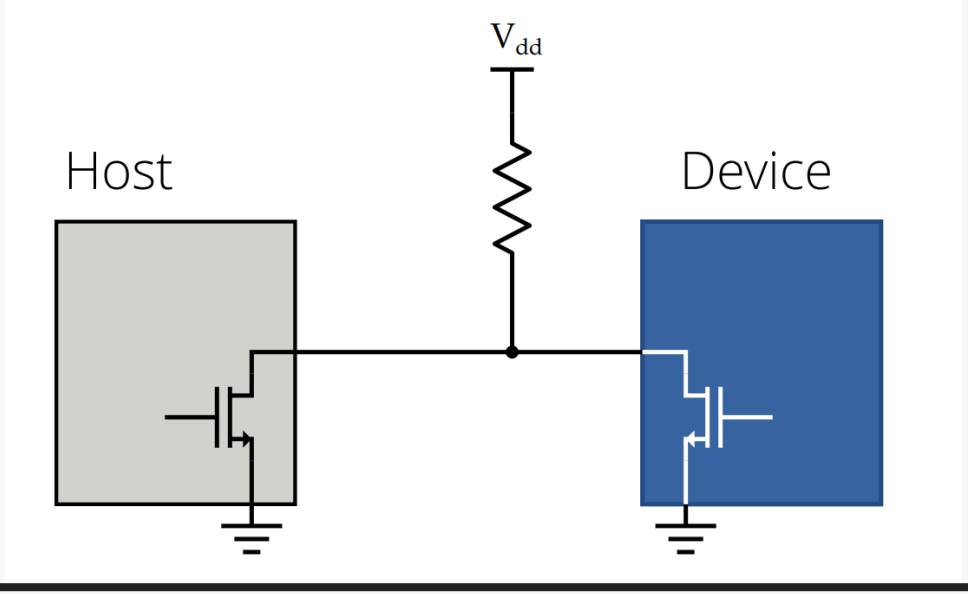
The LOW state wins if any device pulls the line down.

Open-Drain: A Close-Up

With **open-drain outputs**, each device behaves like a switch to ground.

- **Switch open:** no one pulls LOW → resistor produces **HIGH (1)**.
- **Any switch closed:** line is connected to ground → **LOW (0)**.

Devices never actively drive the line HIGH. -- To avoid shorts

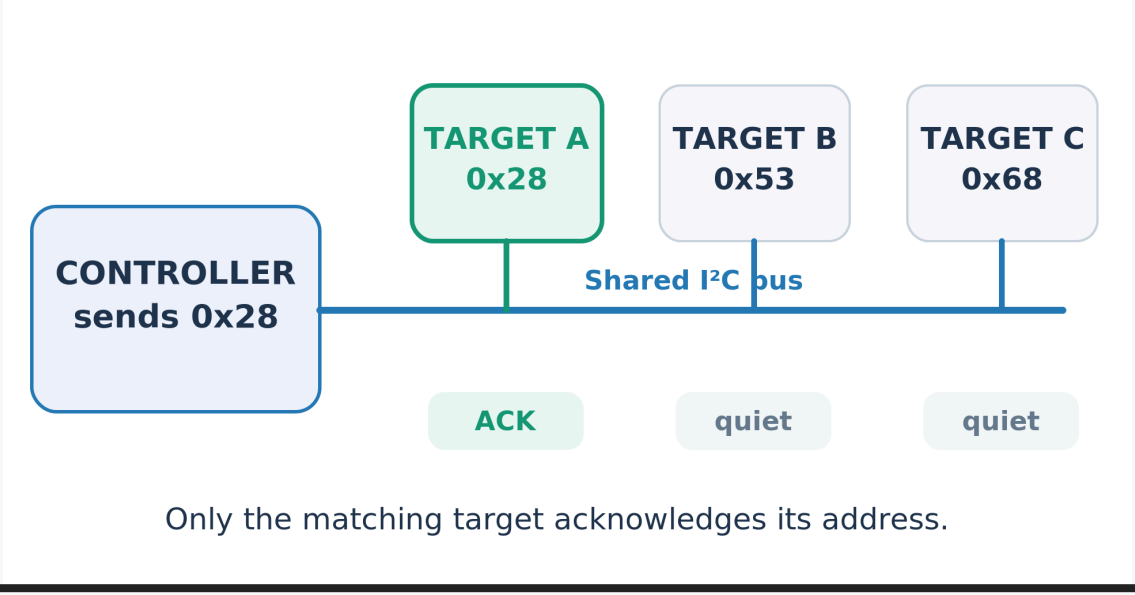


I²C Addressing: Which Sensor?

All targets listen on the same SDA/SCL lines.

- Each target has a **device address**, commonly **7 bits**.
- The controller sends the address at the beginning of a transfer.
- The matching target responds with an **ACK**.

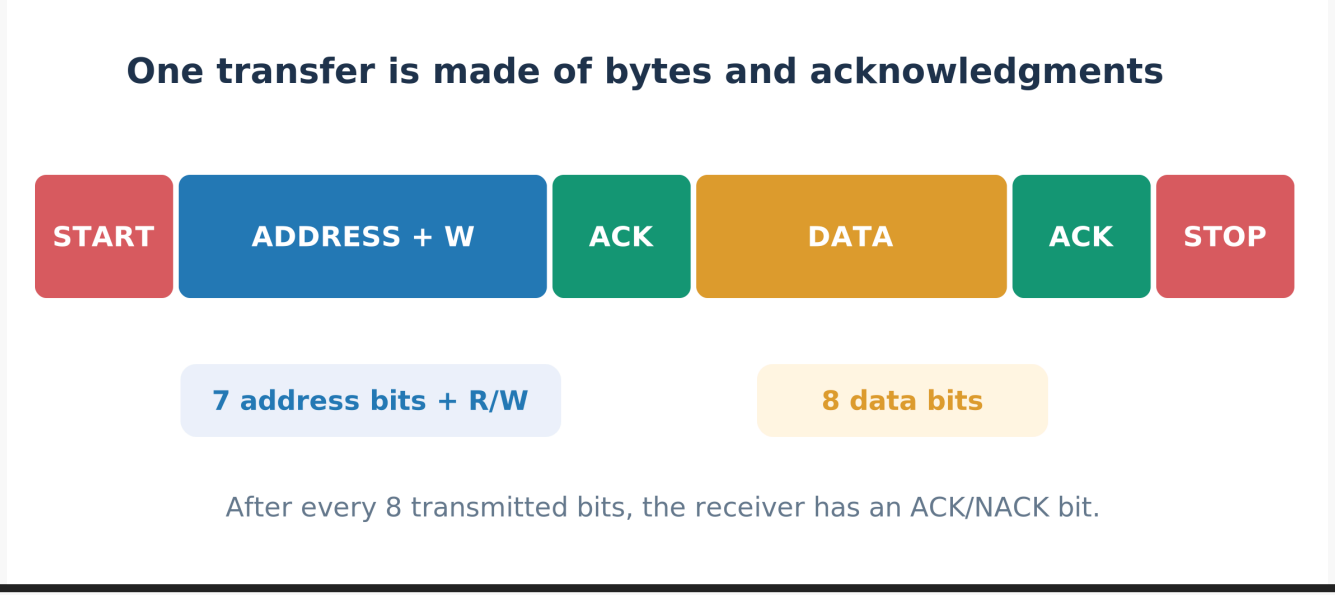
Example: send address `0x28` → only the device configured as `0x28` responds.



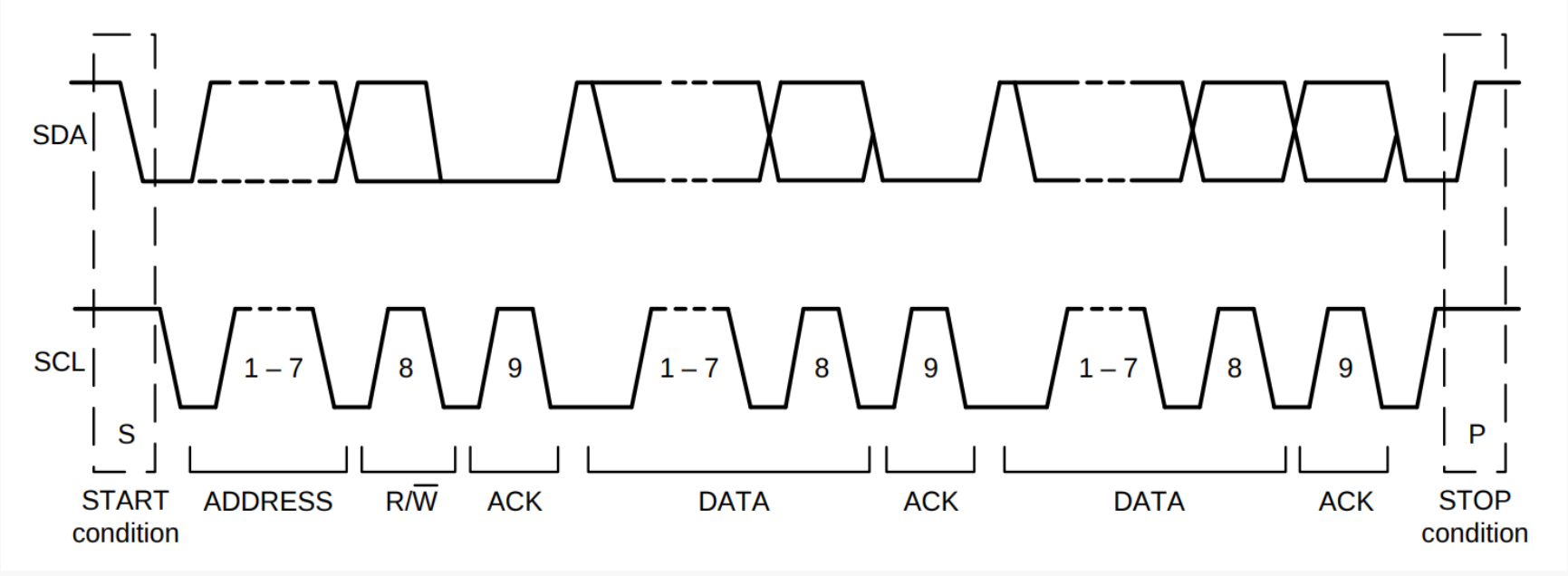
An I²C Transaction: The Big Picture

A transfer is a sequence of **START → address → data → STOP**.

- The address includes the **read/write direction**.
- Data is transmitted in **8-bit bytes**.
- After each byte, the receiver signals **ACK* on the ninth clock pulse.
  - ACK (Acknowledgment): The receiver confirms that a byte was received (SDA = 0).



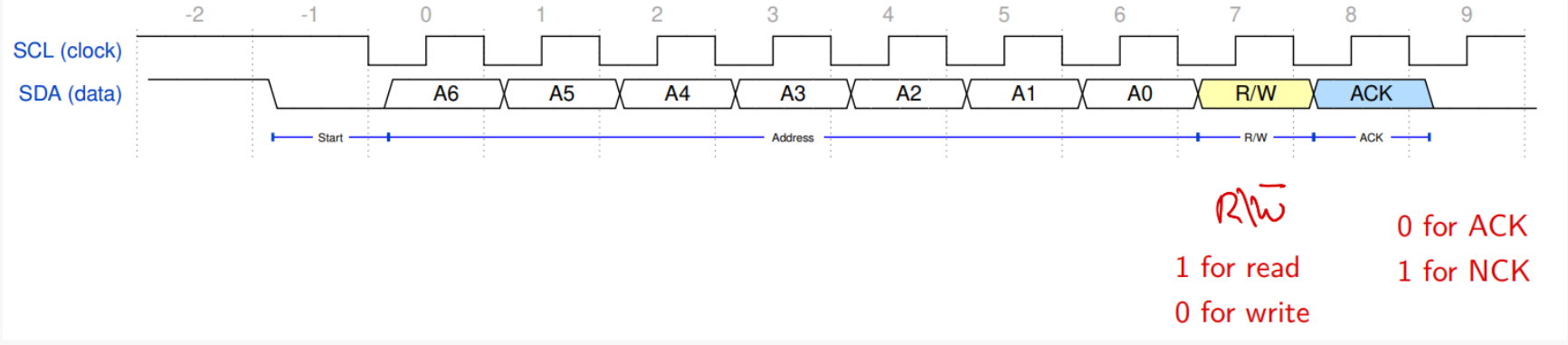
An I²C Transaction: The Big Picture



Address transmission

The first byte after START contains:

- **7 address bits**: which target?
- **1 R/W bit**: `0` = write, `1` = read.
- Then an **ACK** bit from the selected target.



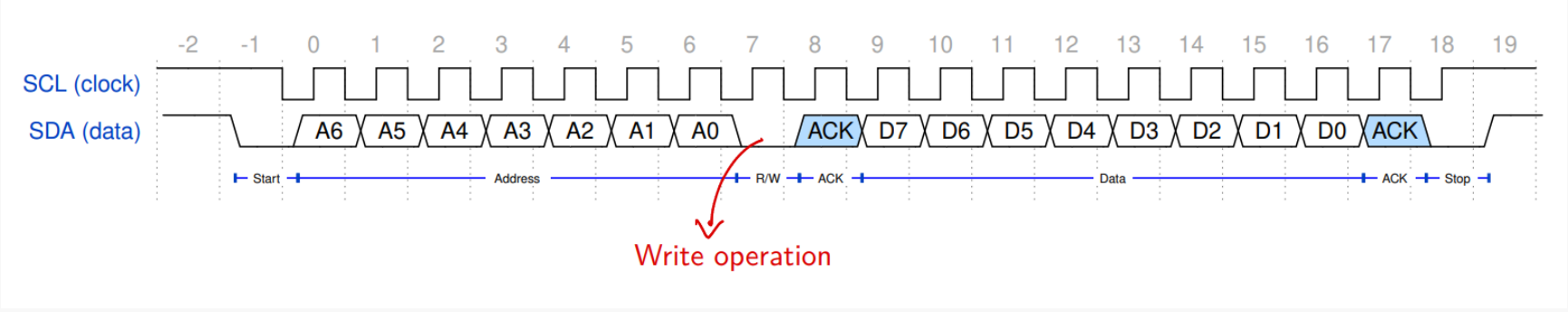
Write Operation: Controller → Sensor

To **write** data, the controller:

1. Sends **START** and the target address with **W = 0**.
2. Sends one or more **data bytes**.
3. Ends with **STOP**.

The **target acknowledges** the address and each received byte.

Used mainly for setting configurations on the target.



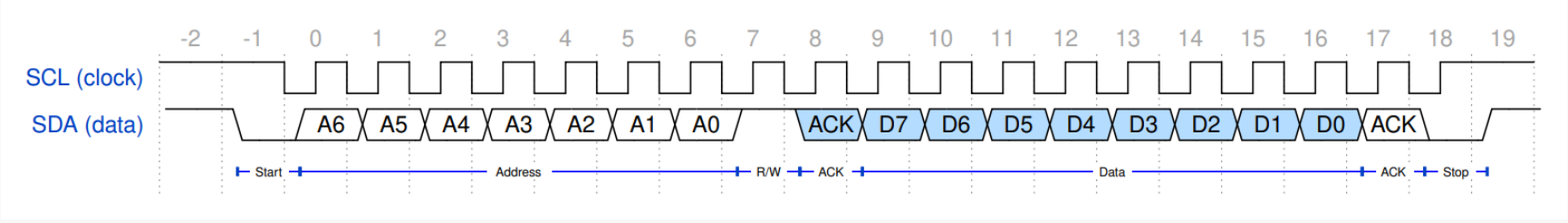
The signal of the sensor id highlighted in blue.

Read Operation: Sensor → Controller

To **read** data, the controller:

1. Sends **START** and the target address with **R = 1**.
2. Continues to provide the **clock** while the sensor transmits bytes.
3. Ends with **STOP**.

The controller controls the transfer; the sensor provides the data.



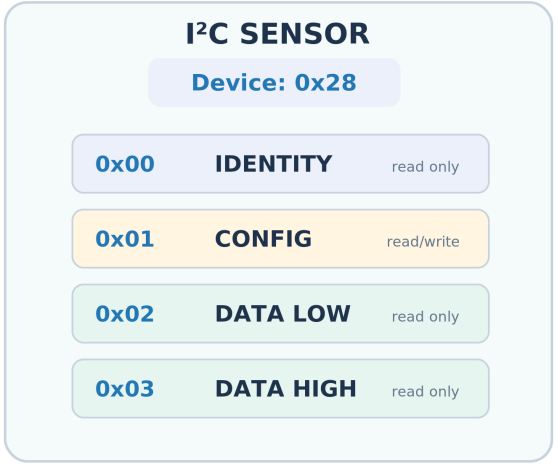
The signal of the sensor id highlighted in blue.

Registers: The Drawers Inside a Sensor

Many I²C sensors organize their information into **registers**: small locations identified by a **register address**.

Think of a sensor as a **cabinet with drawers**:

- **Identity register:** identifies the device / model or chip.
- **Configuration register:** controls settings.
- **Data registers:** hold measurement results.



Illustrative register map – not a specific sensor

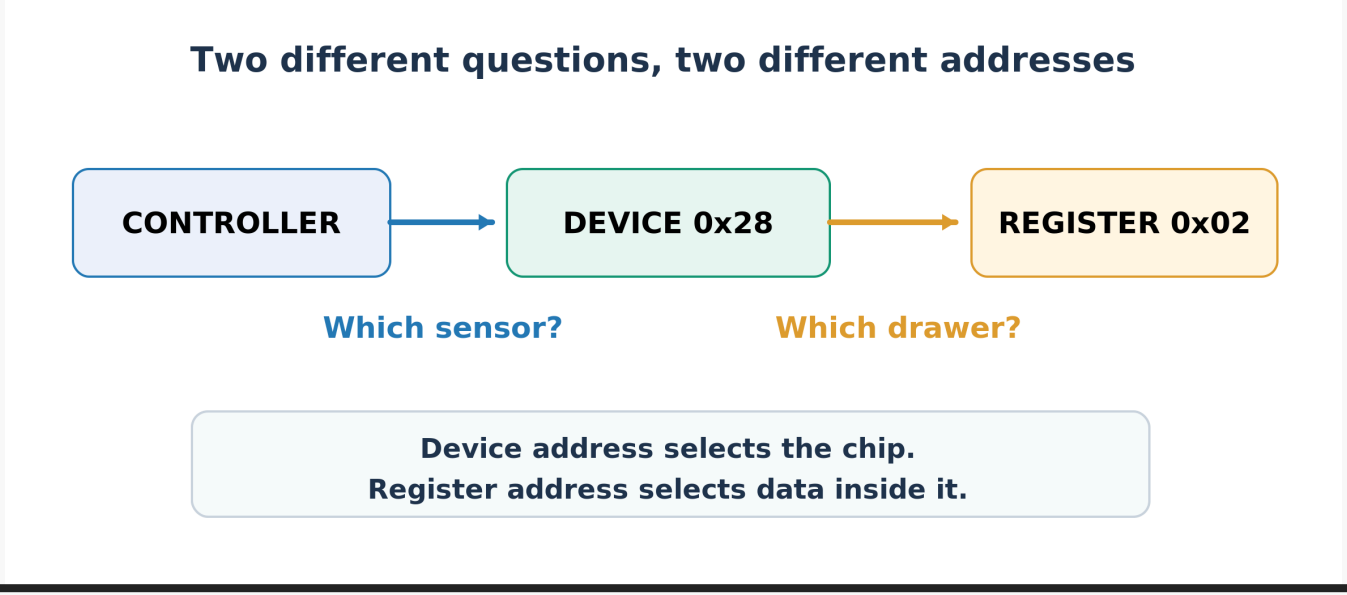
Some drawers can be **read**, others can also be **written**.

Two Addresses: Device and Register

To access a register, the controller must answer **two questions**:

1. **Device address:** *Which sensor am I talking to?*
2. **Register address:** *Which drawer inside that sensor?*

Example: device `0x28`, register `0x02` → read the data stored in that register.

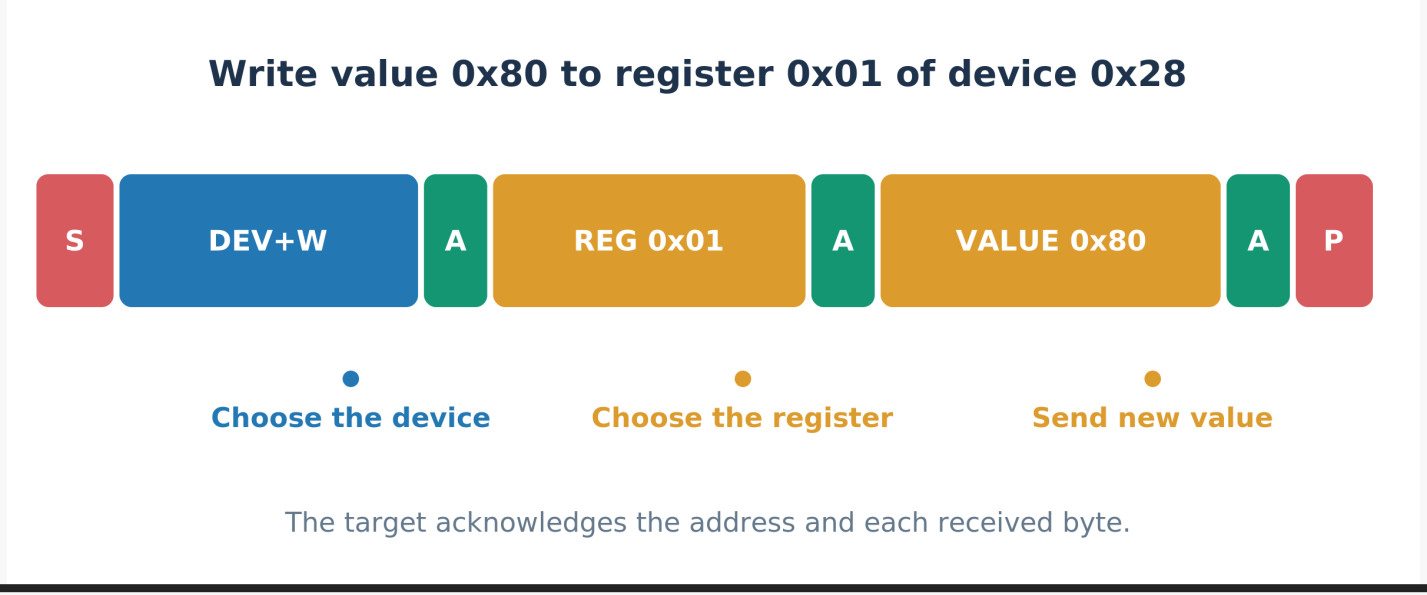


Writing to a Sensor Register

A common **register write** sends:

1. **Device address + W** → select the sensor.
2. **Register address** → choose the drawer.
3. **New data** → update its contents.

Example: write value `0x80` into register `0x01` of device `0x28`.

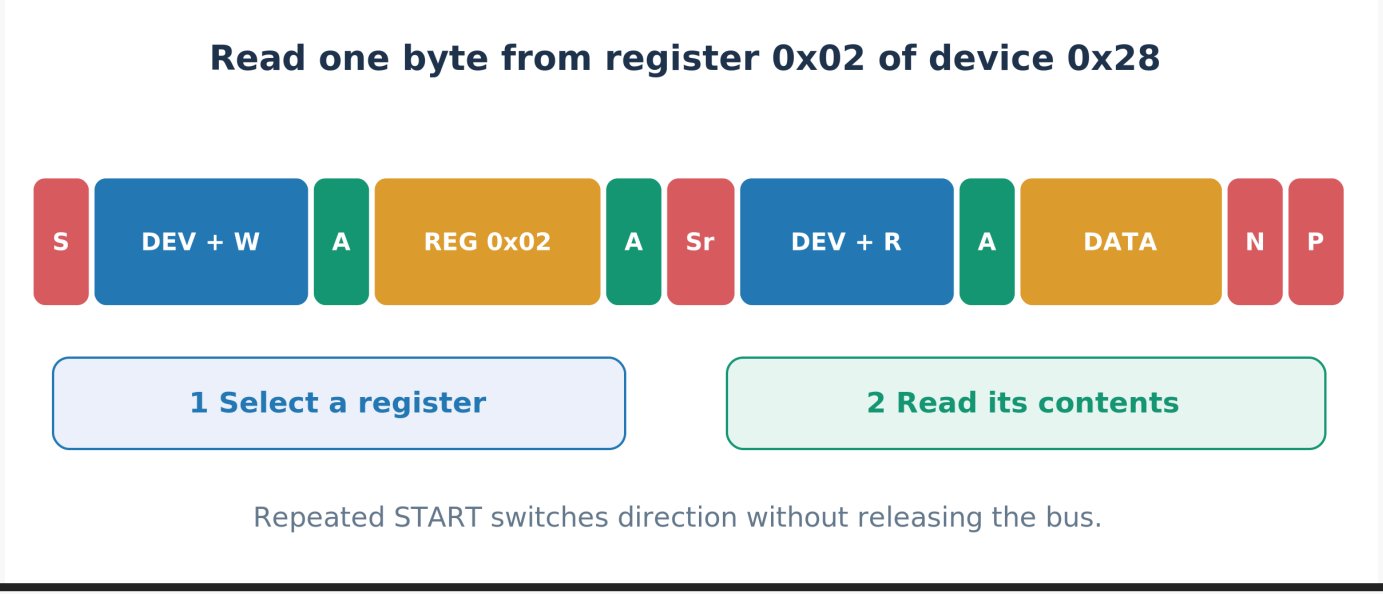


Typical Register Read: Select, Then Read

Reading a specific register usually takes **two phases**:

1. **Write phase:** send the device address and the **register address**.
2. **Read phase:** send a **repeated START (Sr)**, address the device for reading, and receive the data.

The repeated START changes direction without ending the transfer.



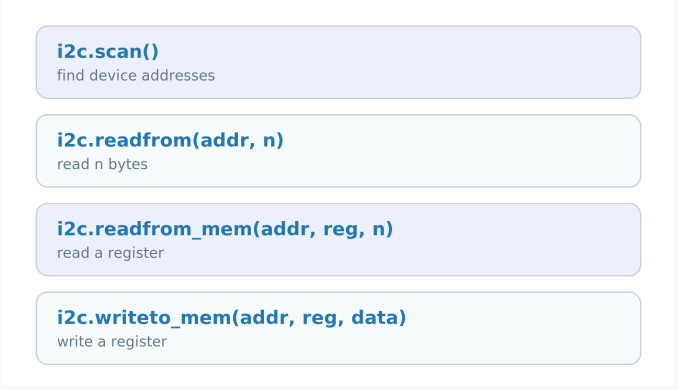
From I²C Transactions to MicroPython

MicroPython handles the low-level **START, ACK and STOP** signaling for us.

Example — read two bytes from a register:

```
raw = i2c.readfrom_mem(0x28, 0x02, 2)
```

- `0x28` → **which sensor**.
- `0x02` → **which register**.
- `2` → **how many bytes**.



Use the datasheet to interpret the returned bytes.