جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

أكاديمية كاوست
KAUST ACADEMY

Internet of Things (IoT)

DAY 3 Reacting to the world: motion, colour, and a display

August 2 to 6, 2026

KGSP Summer Enrichment Program 2026
KAUST Academy, KAUST

Salah Abdeljabar
salah.abdeljabar@kaust.edu.sa

🗺️ On Today's Agenda

1. **Common sensors**: a tour of what we can measure, and how two of them work
2. 🔧 **Project 4**: motion triggers a buzzer, timed without blocking
3. **Commands**: sending instructions back down to a device
4. 🔧 **Project 6**: a browser colour picker mixes a colour on an RGB LED
5. **How sensors talk to the board**: I²C, SPI and UART
6. 🔧 **Project 10**: text and scrolling messages on an I²C OLED
7. 🔧 **Put it together**: motion, colour and the display in one sketch

Tonight's homework: create your Ubidots and Telegram accounts, we need them tomorrow.

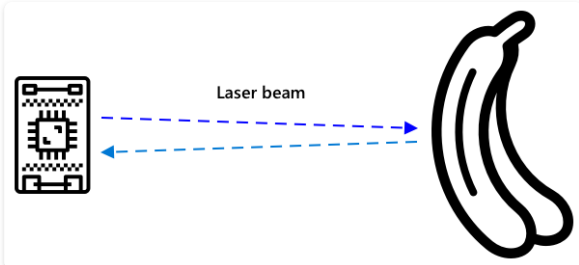
Common Sensors 🔎

A tour of what IoT devices can measure

The Sensor Catalog

- 🌡️ **Temperature**: air or immersion (often + pressure & humidity)
- 🔘 **Button / switch**: pressed or not
- 💡 **Light**: visible, UV, or IR
- 📷 **Camera**: photo or video
- 📈 **Accelerometer**: movement in 3 axes
- 🎤 **Microphone**: sound level or direction
- 📡 **Proximity**: distance to an object

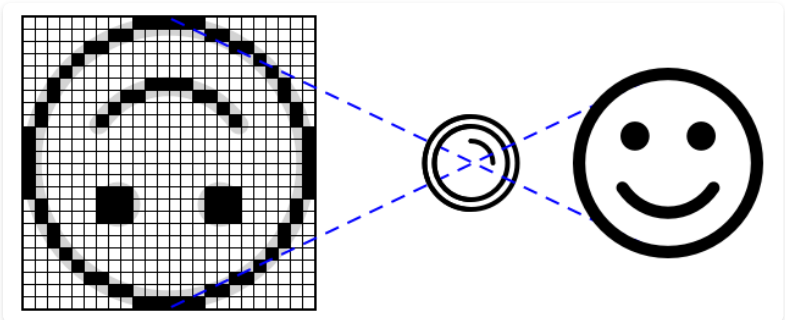
Every sensor converts something physical into an electrical signal.

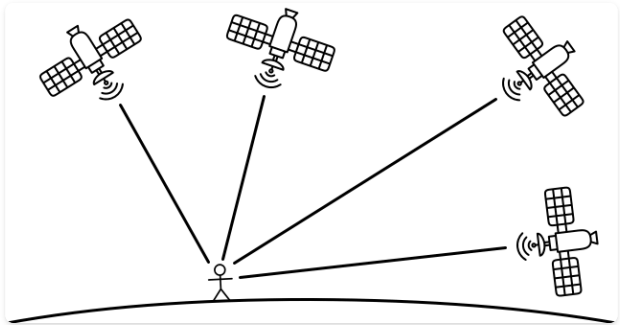


Think

What sensors does your **phone** have?

Inside Two of Them





📷 **Camera**: a lens focuses light onto a grid of pixels, and each pixel reports how much light it caught. The image is just a big array of numbers.

🛰️ **GPS**: the receiver never transmits. It listens to several satellites and works out where it is from how long each signal took to arrive.

> ⓘ **Note**
>
> Both are **digital** sensors: too much data to send as a single voltage, so they do the conversion on board.

PROJECT 4 · SENSOR + TIMERS

PIR Motion Sensor + Buzzer

Detect movement with an HC-SR501 and sound a buzzer: using a non-blocking `millis()` timer.

Builds on: Project 1's active-high digital input, now reading a sensor instead of a button.

⏱️ ~25 min 📊 Beginner 🔖 Prerequisite: Project 1

⬇ Download the sketch

Save `Project_4_ESP32_PIR_Motion_Sensor.ino`, then open it in the Arduino IDE (setup: **Foundation 3**).

Project 4 · PIR Motion Sensor 🚨

Motion trips a buzzer, with non-blocking timing instead of `delay()`.

- Understand what a PIR sensor can and can’t detect
- Wire a 5 V sensor safely from `VIN` rather than `3V3`
- Build a non-blocking timer with `millis()`
- Track state across loop iterations so an event fires once, not continuously

Key pins: PIR `27`, buzzer `26`

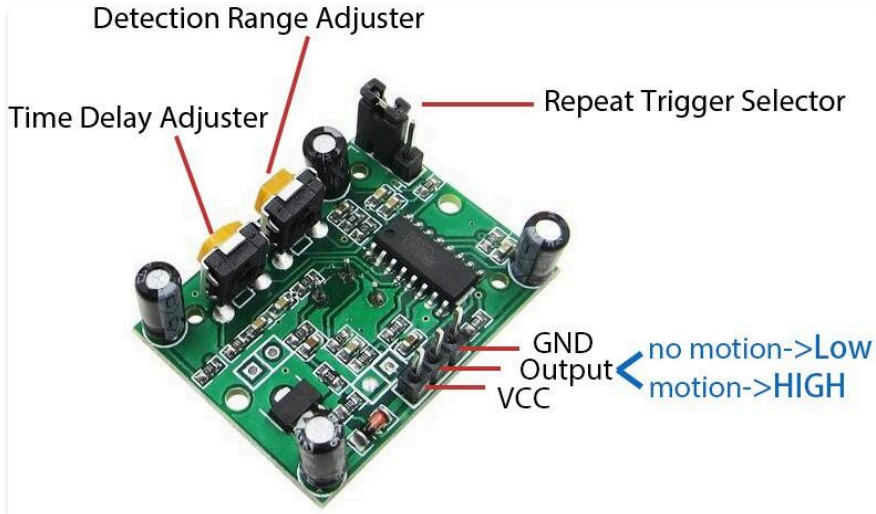
Open the guide ↗

How a PIR Sensor Works

A **Passive Infra-Red** sensor detects the heat that living bodies emit, and triggers on a **change** in that heat, so it reacts to a warm body that is **moving**.

- OUT goes **HIGH** on motion, **LOW** otherwise: read it like a button
- A person standing perfectly still may *not* register
- A cold moving object (a robot, a car) is not seen
- Two trimmers set sensitivity and hold time

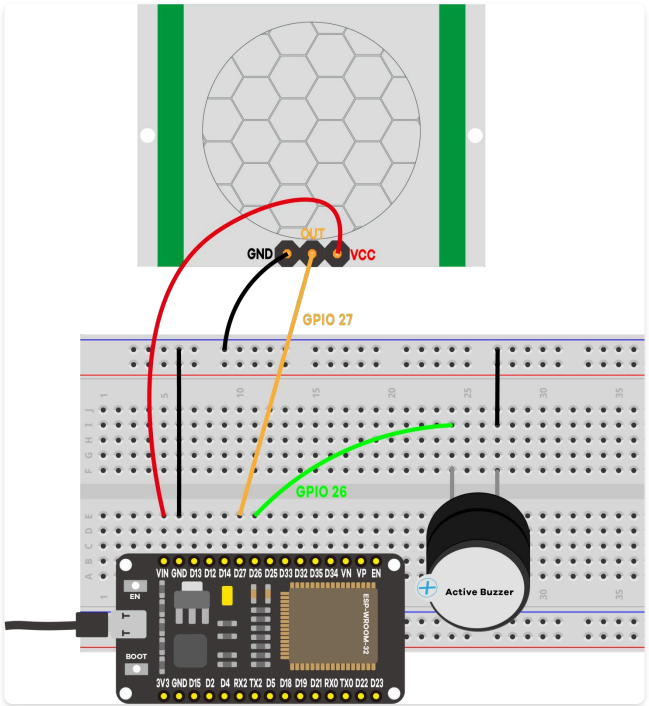
> ⚠ **Warning**
>
> After power-on the PIR needs **30 to 60 s** to warm up before it reads reliably.



Wiring, and Why VIN

From	To
PIR **VCC**	**VIN** (5 V)
PIR **GND**	**GND**
PIR **OUT**	**GPIO 27**
Buzzer **+**	**GPIO 26**
Buzzer **–**	**GND**

> **Note**
>
> The HC-SR501 runs on **5 V**, taken from `VIN` while the board is USB-powered. Its OUT signal is still **3.3 V-safe** to feed back into a GPIO.



Non-Blocking Timing: millis(), not delay()

The buzzer holds on for about 3 s after the last motion, without ever freezing the loop.

```
void loop() {
  bool motion = (digitalRead(motionSensor) == HIGH);
  if (motion) {
    lastMotionTime = millis();                // remember "now"
    if (!alarmActive) { digitalWrite(buzzerPin, HIGH); alarmActive = true; }
  } else if (alarmActive && millis() - lastMotionTime >= holdTime) {
    digitalWrite(buzzerPin, LOW); alarmActive = false;   // hold elapsed
  }
}
```

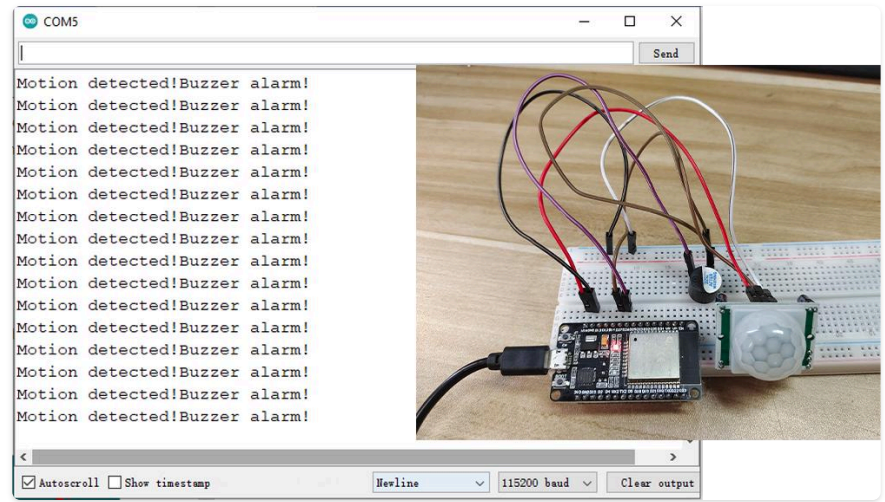
- `millis()` is milliseconds since boot, a clock that never stops (store it in an `unsigned long`).
- Save a timestamp and subtract later: `millis() - startedAt >= interval` is the **non-blocking timer**.
- The `alarmActive` flag fires the event **once** on the rising edge, not on every pass.

It Sounds

Wave a hand in front of the sensor and the buzzer sounds with a `MOTION detected` line. Stay still and it stops after about 3 s. Fresh motion pushes the timer forward and extends the alarm.

What you learned:

- A PIR sees moving heat, not stillness or cold, and needs a warm-up
- 5 V modules run from **VIN**, with 3.3 V logic back to the GPIO
- `if (millis() - startedAt >= interval)` is the timer to reuse (Projects 8, 11)
- A state flag separates a **new event** from an ongoing condition



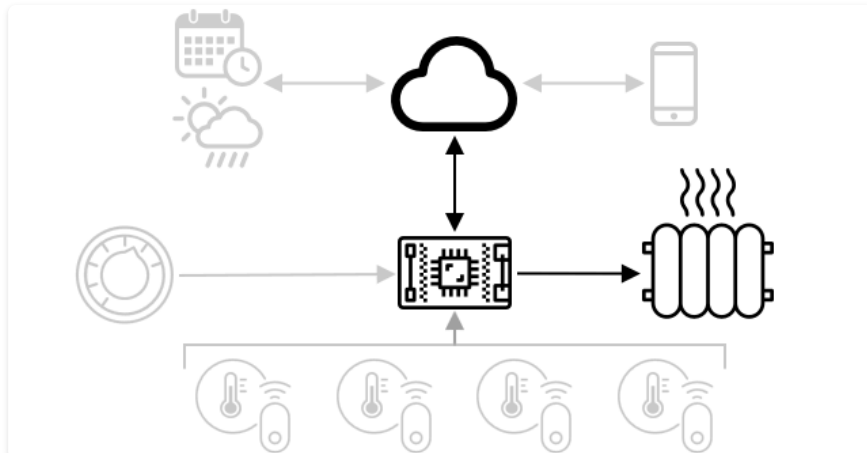
Commands 

Controlling devices from the cloud

Cloud → Device

A **command** is a message from the cloud telling a device to act: drive an actuator, reboot, or send extra data.

- Server checks telemetry → publishes to `/<id>/commands`
- Device subscribes and acts
- e.g. light < 300 → `{ "led_on": true }` → LED on



When the Device is Offline

What if a command can’t be delivered? **It depends on the command.**

Latest wins → discard

A thermostat **setpoint**: a newer command overrides the old one, so drop the stale ones.

Order matters → queue

A **robot arm** ("up", then "grab"): commands must be **queued and replayed** in sequence.

Meme check

The "bricked car", trapped underground with no signal: always design for lost connectivity!

PROJECT 6 · WI-FI / WEB

RGB LED Web Server: Color Picker

Pick any color in the browser; the ESP32 mixes it on an RGB LED with three PWM outputs.

Builds on: Project 3's PWM and Project 5's web server, combined rather than replaced.

⏱️ ~30 min 📊 Intermediate 🔖 Prerequisite: Project 3 & 5

⬇ Download the sketch

Save `Project_6_RGB_LED_Web_Server.ino`, then open it in the Arduino IDE (setup: **Foundation 3**). Remember

Project 6 · RGB LED Web Server 🎨

A browser colour picker mixes a colour on an RGB LED over Wi-Fi.

- See how one RGB package holds three separate LEDs
- Mix colour with three PWM duty cycles, the same way a screen does
- Drive three PWM outputs at once with `ledcAttach()`
- Parse three numbers out of a request URL

Key pins: R/G/B `13`/`12`/`14`

Open the guide ↗

The Big Picture

This project **combines** two earlier ones: Project 3's PWM and Project 5's web server.

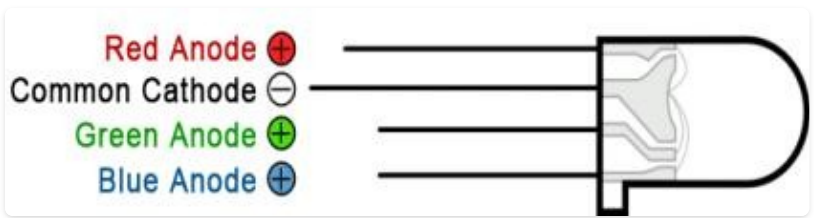
- A browser **color picker** sends the chosen color to the board
- The ESP32 mixes it with **three PWM outputs**, one per color
- Nothing new to learn about hosting a page: it is Project 5's server

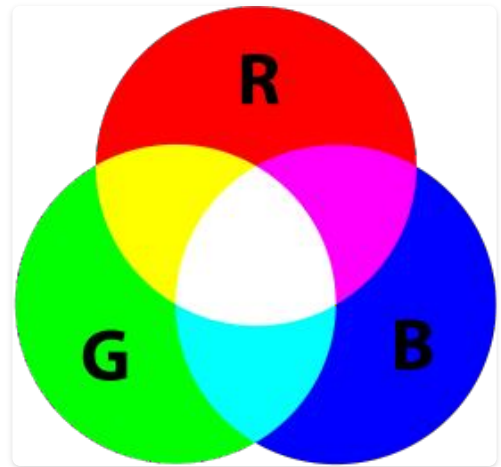
Most real devices are built this way, by joining pieces you already have.

An RGB LED and Additive Color

An RGB LED is really **three LEDs in one**: red, green, blue. The kit's are **common cathode**: one shared **–** leg to GND, and each color has its own anode, resistor, and PWM pin.

- Red + Green = Yellow, Green + Blue = Cyan, Red + Blue = Magenta
- All three at full = White, all off = dark





Your eye blends the three elements into one color.

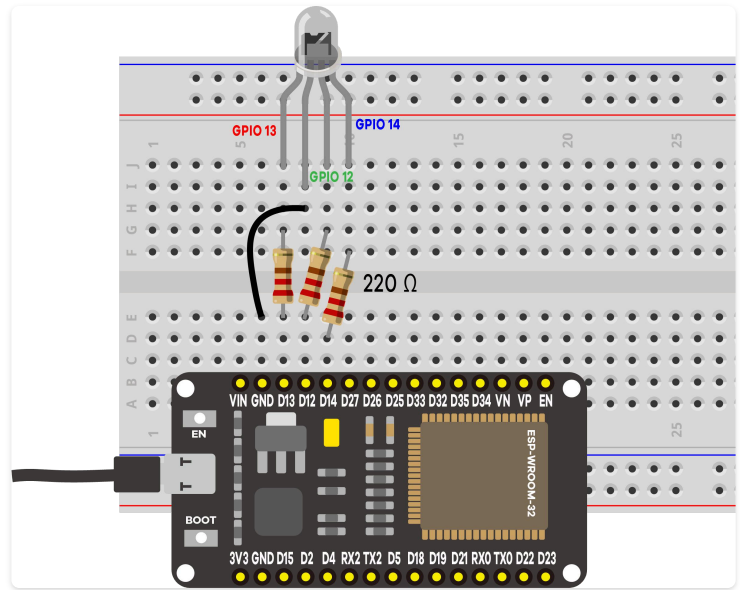
Three PWM Outputs

It is Project 3's PWM, done **three times**, once per color, in `setup()`:

```
ledcAttach(redPin,   5000, 8);   // GPIO 13
ledcAttach(greenPin, 5000, 8);   // GPIO 12
ledcAttach(bluePin,  5000, 8);   // GPIO 14
```

- **Red** anode → 220 Ω → **GPIO 13**
- **Green** anode → 220 Ω → **GPIO 12**
- **Blue** anode → 220 Ω → **GPIO 14**
- Common cathode → **GND**

The PWM controller is hardware, so it holds the color while the processor serves pages.



A URL Can Carry Data, Not Just a Command

The picker sends a request like `/?r201g32b255&`. A small hand-rolled parser slices out each number:

```
pos1 = header.indexOf('r'); pos2 = header.indexOf('g');
pos3 = header.indexOf('b'); pos4 = header.indexOf('&');
redString   = header.substring(pos1 + 1, pos2);   // "201"
greenString = header.substring(pos2 + 1, pos3);   // "32"
blueString  = header.substring(pos3 + 1, pos4);   // "255"
ledcWrite(redPin,   redString.toInt());           // 0..255 brightness
ledcWrite(greenPin, greenString.toInt());
```

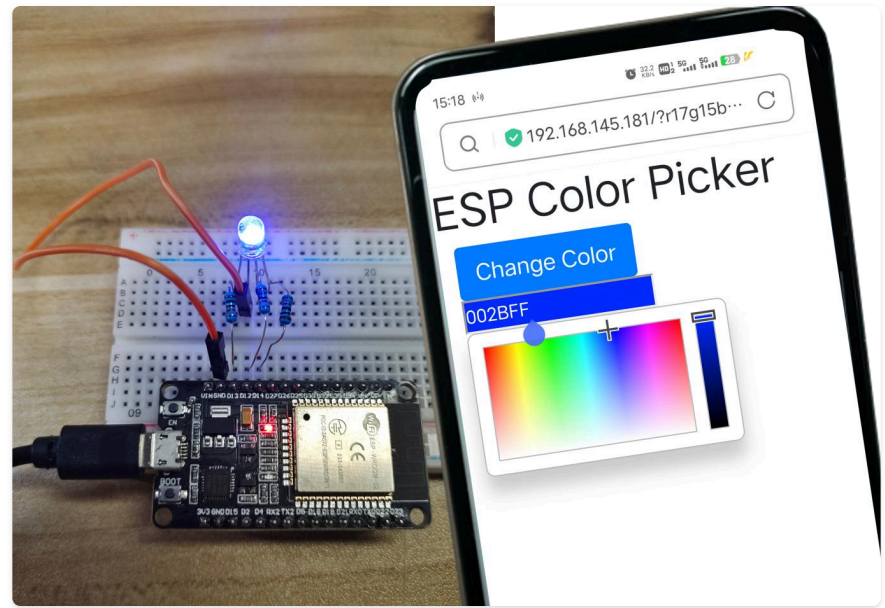
- `r`, `g`, `b`, `&` are **delimiters**; each value is whatever sits between two of them.
- `substring()` copies those digits, `toInt()` turns `"201"` into `201`.
- **8-bit** gives 0 to 255 per color, exactly the picker's output: $256^3 \approx$ **16.7 million** colors.

It Mixes Any Color

Pick a color, press **Change Color**, and the LED matches it. Pick black to turn it off.

What you learned:

- An RGB LED is three LEDs sharing one leg; common cathode goes to **GND**
- Color mixing is **additive**, and your eye does the blending
- A URL can carry **data**, pulled out with `indexOf()` and `substring()`
- Hardware PWM runs independently of your code



How Sensors Talk to the Board 🔌

The buses behind every digital sensor

Three Ways to Wire a Peripheral

A digital sensor or actuator sends 0s and 1s, but *over what*? Nearly every board uses one of three buses.

I²C	SPI	UART
Two wires, **many** devices, each with an address. Slower, but cheap on pins.	Faster, **full duplex**, one extra select wire per device.	Two devices, **no clock**: both sides just agree on a speed.

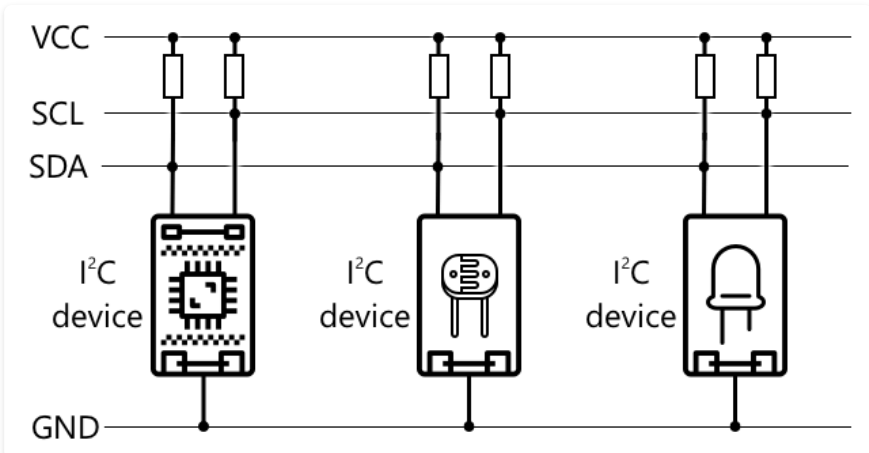
Tip

You rarely wire these by hand: a library exposes `read()` and hides the bus. But knowing which bus a sensor uses tells you how many you can connect.

I²C: One Bus, Many Devices

Inter-Integrated Circuit shares just two lines across every device:

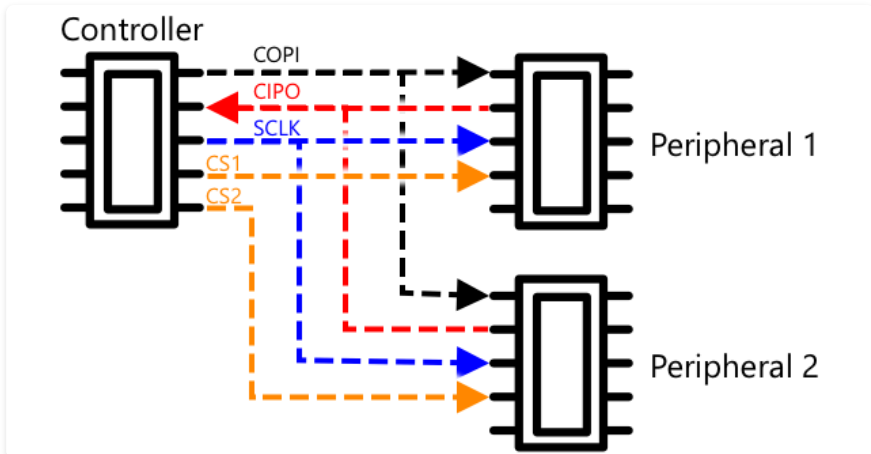
- **SDA** carries the data, **SCL** the clock
- Each device has an **address**, so the controller can name who it wants
- Add sensors without adding pins
- Typical of temperature, humidity, and light sensors



SPI: Fast and Full Duplex

Serial Peripheral Interface trades pins for speed:

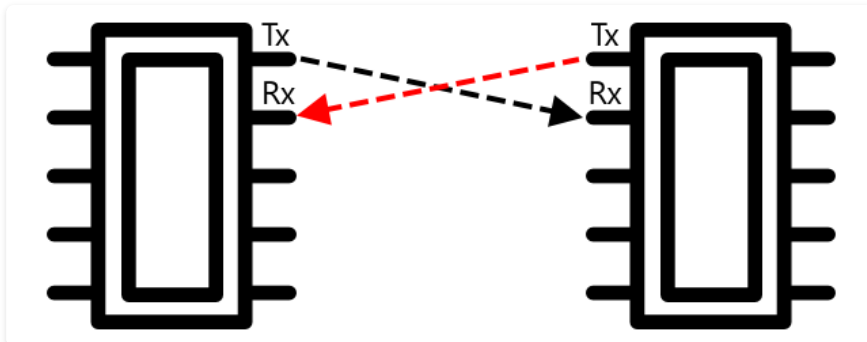
- **COPI** and **CIPO** send in both directions at once
- **SCLK** keeps both sides in step
- A **chip select** line per peripheral picks who is listening
- Good for displays, SD cards, and cameras



UART: A Direct Conversation

Universal Asynchronous Receiver-Transmitter connects exactly two devices:

- **Tx** on one side wires to **Rx** on the other, and back again
- **Asynchronous**: no clock line, so both must agree on a **baud rate**
- The serial monitor on your laptop is a UART
- Common for GPS and radio modules



PROJECT 10 · DISPLAY

OLED Display (SSD1306)

Show text on a 0.96″ 128×64 I²C screen: a building block for local, phone-free output.

Builds on: Foundation 1's I²C introduction, now driving a real display instead of just naming the bus.

⏱️ ~25 min 📊 Intermediate 🔖 Prerequisite: Foundation 2

⬇ Download the sketch

Save `Project_10_ESP32_OLED_Display.ino`, then open it in the Arduino IDE (setup: **Foundation 3**). Needs the Adafruit SSD1306 + GFX libraries.

Project 10 · OLED Display

Text and scrolling messages on a 128×64 I²C OLED.

- Understand I²C: a two-wire bus, many devices, each with its own address
- Drive a display through a chip library and a graphics library
- Learn the buffer-then-push pattern every graphical display uses
- Check that hardware actually started before drawing to it

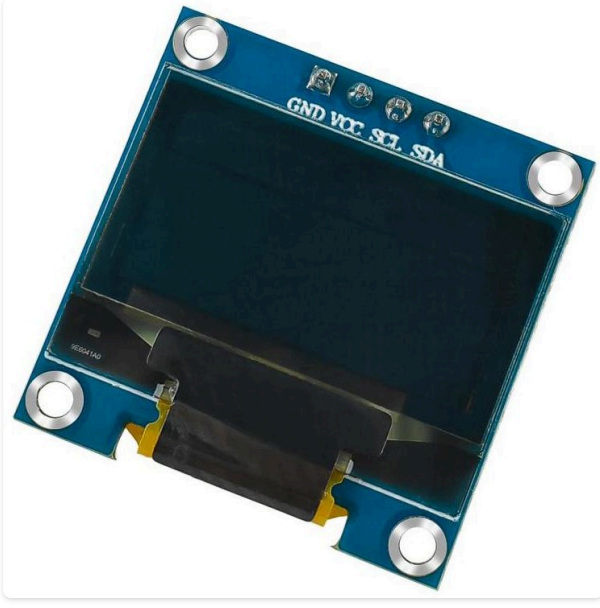
Key pins: SDA `21`, SCL `22`

I²C and the OLED

The SSD1306 is a crisp **128×64** monochrome OLED. It talks over **I²C**, a two-wire bus: **SDA** (data) and **SCL** (clock).

- Default ESP32 I²C pins: **GPIO 21 (SDA)**, **GPIO 22 (SCL)**
- Each I²C device has an **address**; this one is **0x3C**
- VCC → **3.3 V**, GND → **GND**: four wires, no resistors

Adding a second I²C part costs no extra pins, just a different address.



Layered Libraries, Buffer Then Push

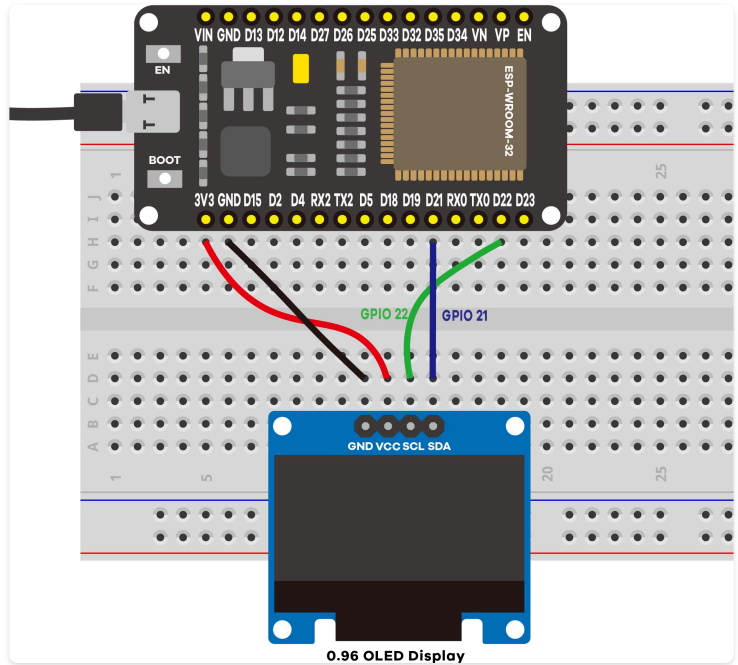
```
Adafruit_SSD1306 display(128, 64, &Wire, -1);  // size, I2C bus, no reset pin

display.clearDisplay();       // edits the buffer in RAM, not the screen
display.setTextSize(2);
display.setCursor(0, 30);     // (0, 0) is the top-left corner
display.println("LAFVIN");    // still only in RAM
display.display();            // <-- pushes the frame over I2C
```

- **Layers:** `Wire` moves bytes, `Adafruit_SSD1306` speaks the chip's commands, `Adafruit_GFX` draws text and shapes, so the same `println()` you know works on a screen.
- Every drawing call edits a 128×64 **buffer** in RAM; only `display()` sends it.
- A forgotten `display()` looks exactly like a dead screen: the most common OLED mistake.

Check It Started, Let the Chip Scroll

```
if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
  Serial.println("not found: check");
  Serial.println("SDA->21, SCL->22, 0x3C");
  for (;;);   // stop: nothing to do
}
display.startscrollright(0x00, 0x0F);
```

- `begin()` returns `false` if nothing answers at **0x3C**, which is what a loose wire looks like
- Three lines turn a silent freeze into a message naming the wires to check
- Scrolling runs on the **display chip**, not the CPU

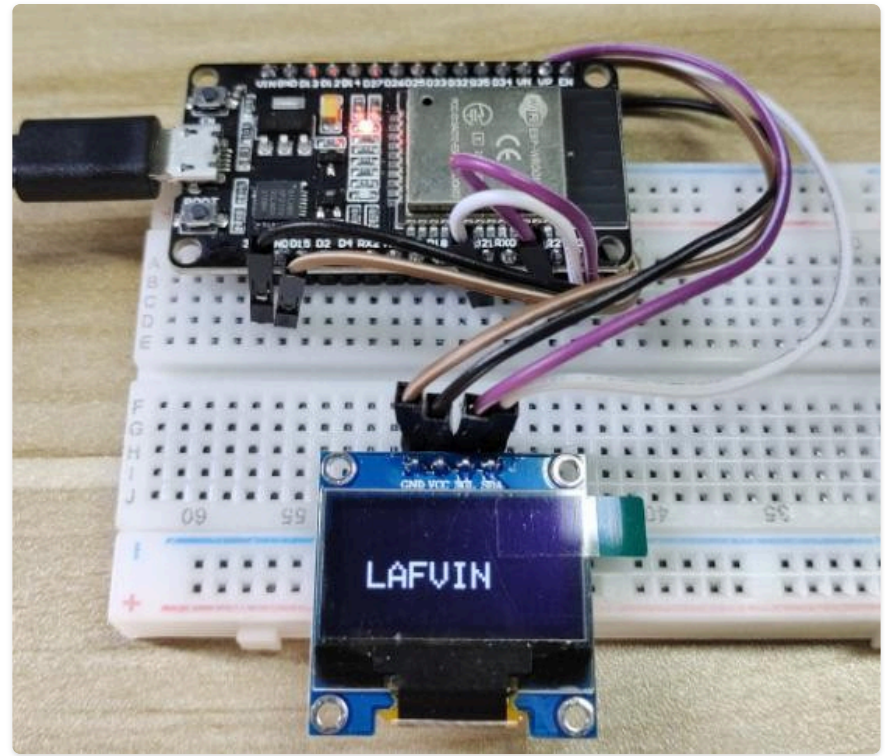


Local Output

The text appears, then scrolls right and left on a loop, all with no phone and no network.

What you learned:

- **I²C** carries many devices on two wires, each with an address
- A display is driven by **layers**, from raw bytes up to text
- Graphics work on a **buffer**; `display()` pushes the frame
- A building block: feed it the DHT11 reading or the ESP32's IP



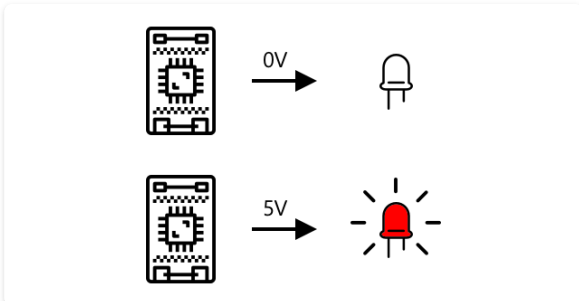
Common Actuators 🔧

A tour of how IoT devices affect the world

The Actuator Catalog

- 💡 **LED**: emit light
- 🔊 **Speaker / buzzer**: sound, from beeps to music
- ⚙️ **Stepper motor**: precise, defined rotation
- 🔀 **Relay**: a small signal switches a big circuit
- 🖥️ **Screen**: from simple segments to HD video

Actuators are the output half of IoT, turning signals into action.



Think

What actuators does your **phone** have?

REFERENCE · COMPONENTS

Components Glossary

Every part in the kit: what it is, how to recognize it, whether direction matters, and where it's used.

> **Polarity key:** 🔴 **polarized**: direction matters, can be damaged if reversed · ○ **not polarized**: either way is fine.

Reference · Components Glossary 🧩

Every part in the kit: how to identify it, whether it's polarized, and which project uses it.

- A photo and description of each component in the kit
- Polarity called out where it matters
- A pointer to the project that first introduces each part

Open the guide ↗

Thank you!

For spending the day building the Internet of Things with us.

Questions? Ask now, or write to me any time.

Slides, labs, and resources

KGSP Summer Enrichment Program 2026
KAUST Academy, KAUST

Salah Abdeljabar
salah.abdeljabar@kaust.edu.sa