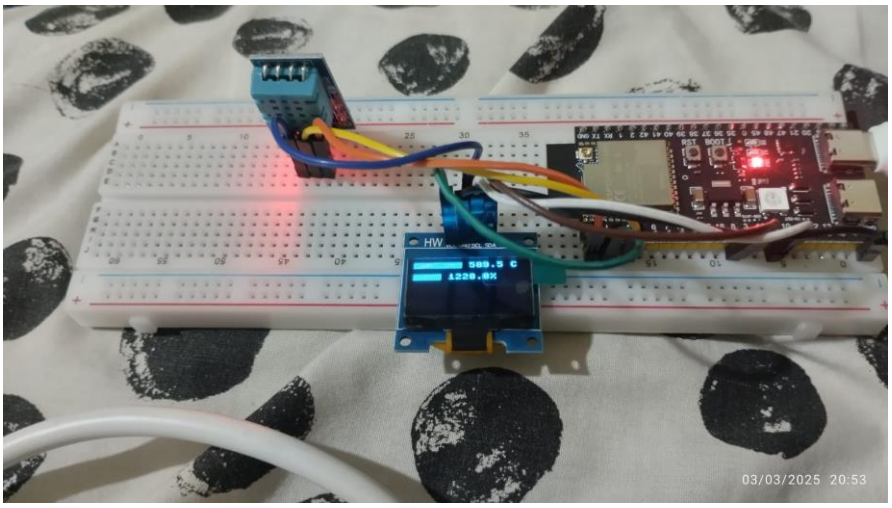
LAB -3-Home-Task

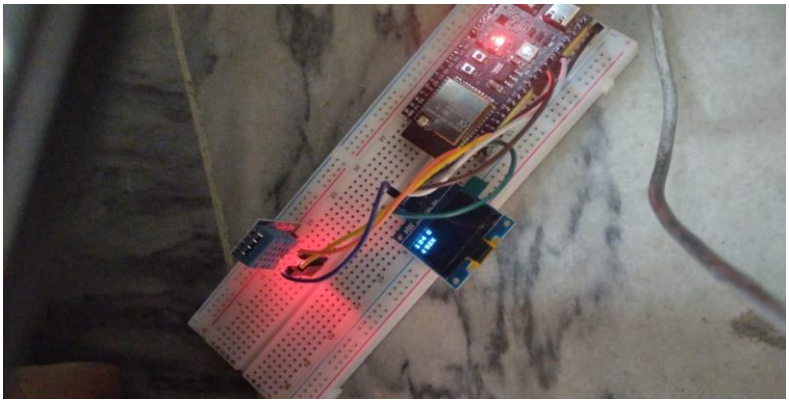
📜 **Task 1:** Displaying Temperature & Humidity on OLED

- Print the values of temperature and humidity on the OLED display.
- Try adding emojis for temperature and humidity to see if the OLED supports them.
- Blow on the sensor and observe whether it detects minor changes in temperature and humidity.

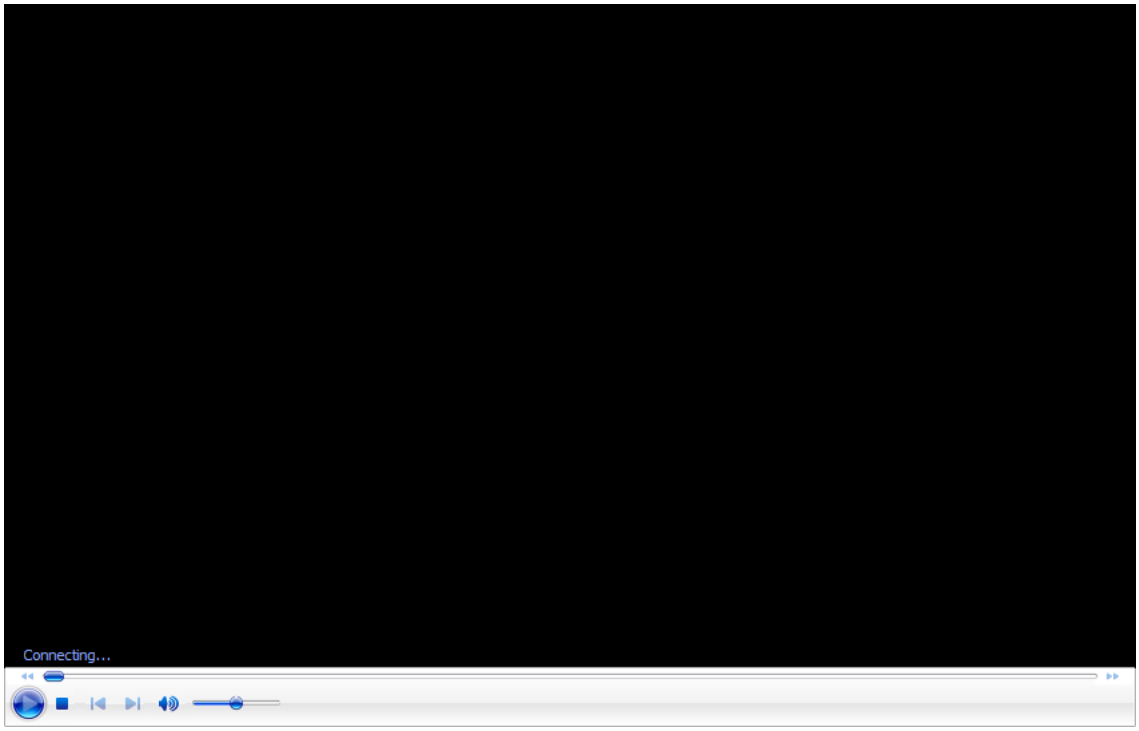
1.1 Print the values of temperature and humidity on the OLED display.



1.2 Try adding emojis for temperature and humidity to see if the OLED supports them.



1.3 Blow on the sensor and observe whether it detects minor changes in temperature and humidity.



📜 **Task 2:** Running the Code Without Interrupt

- Run the same functionality of the lab work without interrupt & log your observations. Also state, what happened before and after interrupt?

If the program runs without an interruption (button press) . The **temperature and humidity** are continuously read and displayed on the **OLED screen** without any interruptions. The **OLED remains ON** and keeps updating with new readings. No toggling of the display occurs, as the button is never pressed.

Before Interrupt:

The ESP32-S3 continuously reads temperature and humidity from the DHT22 sensor and displays the values on the OLED screen. The OLED remains ON and updates every second without any interruptions.

After Interrupt:

When the button is pressed, an interruption is triggered, toggling the OLED display ON/OFF. A debounce timer prevents multiple triggers. Without the interruption, the display stays ON and updates normally, but with the interrupt, the user can control the display state.

📜 **Task 3:** Understanding Debounce Issue

- What is a debounce issue and why we get rid of it?
- In which applications/domains, debounce issue can be threatening if not resolved in the system?
- Why debounce occurs? Is it a compiler error, logical error or micro-controller is cheap?

Activat
Go to Set

3.1 What is a debounce issue and why we get rid of it?

A **debounce issue** occurs when a switch is pressed, and it generates multiple unwanted signals (bounces) due to physical contact vibrations. This can cause a single press to be registered multiple times, leading to unintended behavior. We get rid of it to ensure accurate and reliable input detection, preventing errors in user interactions, especially in embedded systems and real-time applications.

3.2 In which applications/domains, debounce issue can be threatening if not resolved in the system?

Debounce issues can be critical in:

- **Industrial automation** – False triggering of control buttons can cause machine malfunctions.
- **Medical devices** – Incorrect button presses can lead to life-threatening errors.
- **Banking and security systems** – Multiple inputs can cause unauthorized transactions or access issues.
- **Gaming and user interfaces** – Unintended actions can affect user experience.
- **Embedded systems** – Unstable signals can disrupt system logic.

3.3 Why does debounce occur? Is it a compiler error, logical error or micro-controller is cheap?

Debounce occurs due to **mechanical limitations** of physical buttons and switches, not due to a compiler or logical error.

- It is **not a compiler error** because it happens at the hardware level.
- It is **not a logical error** unless the software fails to handle it properly.
- It is **not caused by a cheap microcontroller**, but cheaper components might have worse bouncing behavior.

The issue arises from the natural mechanical behavior of switches, which is why **debouncing techniques** (software or hardware-based) are used to filter out false signals.

📜 **Task 4:** Why Do We Use Interrupts

- Why do we use interrupt?
- How does interrupt lower the processing cost of the micro-controller?

4.1 Why do we use interrupts?

Interrupts allow a microcontroller to respond immediately to important events without continuously checking for them. This improves efficiency by enabling the processor to perform other tasks while waiting for an event, rather than constantly polling for changes.

4.2 How does interrupt lower the processing cost of the micro-controller?

Interrupts reduce CPU workload by eliminating the need for continuous polling. Instead of running loops to check for inputs, the processor can remain idle or execute other tasks, only responding when an interrupt occurs. This saves power and improves overall system performance.