**Automation and IOT Workshop - Instructions**

* **Vishnu Raj**

**(Microgrid And Drives Lab,**

**Dept of EEE,**

**Amrita School of Engineering Bengaluru)**

**Getting Started with Hardware(Embedded Systems In Automation):**

* Understanding micro controllers
  + Definition: A microcontroller is a compact integrated circuit that contains a processor core, memory, and programmable input/output peripherals. It is designed to execute specific tasks and is commonly used in embedded systems.
  + Functionality: Microcontrollers are used to control and manage a variety of electronic devices and systems. They process input data, execute programmed instructions, and produce output results to interact with the environment.
  + Components: Typical components of a microcontroller include a CPU (Central Processing Unit), RAM (Random Access Memory), ROM (Read-Only Memory) or Flash memory for program storage, timers, counters, and input/output ports.
  + Programming: Microcontrollers are programmed using low-level languages like Assembly or high-level languages like C. The program is typically stored in non-volatile memory and is executed by the microcontroller's CPU.
  + Applications: Microcontrollers are widely used in everyday devices such as microwave ovens, washing machines, cars, and more complex systems like robotics, medical devices, and industrial automation.
  + Real-time Operation: Microcontrollers are often designed to operate in real-time, meaning they must respond to inputs and produce outputs within specific time constraints. This makes them suitable for applications where timing is critical.
  + Low Power Consumption: Many microcontrollers are designed to operate on low power, making them suitable for battery-powered devices and other applications where energy efficiency is important.
  + Peripherals: Microcontrollers come with various peripherals, such as GPIO (General Purpose Input/Output) pins, UART (Universal Asynchronous Receiver/Transmitter) for serial communication, ADC (Analog-to-Digital Converter) for reading analog signals, and more.
  + Development Tools: To program and debug microcontrollers, developers use specialized tools such as Integrated Development Environments (IDEs), compilers, and in-circuit emulators.
  + Variety of Architectures: Microcontrollers come in various architectures, such as ARM, AVR, PIC, and 8051. The choice of architecture depends on the specific requirements of the application.

**Getting Started with Arduino IDE:**

**Step 1: click on this link** [**https://docs.arduino.cc/hardware/uno-rev3**](https://docs.arduino.cc/hardware/uno-rev3)

**Step 2: Download and install the latest version of Arduino IDE**

**Step 3: install the library for DS18B20 Sensor “Dallas temperature sensor library ”**

**Files🡪 Examples🡪 Blink**

**Hardware Setup:**

1. **Connect the DS18B20 Sensor:**
   * Connect the **DS18B20** sensor to pin 2 on the Arduino (e.g., A0).
   * Connect a resistor (pull-up resistor) of 10K between the other pin of the sensor and the 3.3V pin on the Arduino.
   * Connect the other end of the sensor to the ground (GND) pin on the Arduino.
2. **Connect the Relay Module:**
   * Connect the control pin of the relay module to one of the digital pins on the Arduino (e.g., D2).
   * Connect the VCC and GND pins of the relay module to the 5V and GND pins on the Arduino.
3. **Connect the Serial Communication:**
   * Connect the Arduino to your computer via USB.

++++++++++++++++++++++++++++++ **Arduino Code example**++++++++++++++++++++++++++++++

const int sensorPin = A0; // Analog pin for NTC sensor

const int relayPin = 2; // Digital pin for relay

void setup() {

Serial.begin(9600);

pinMode(relayPin, OUTPUT);

}

void loop() {

// Read temperature from NTC sensor

int sensorValue = analogRead(sensorPin);

float voltage = sensorValue \* (5.0 / 1023.0);

float temperature = (voltage - 0.5) \* 100.0;

// Control the relay based on temperature

if (temperature > 25.0) {

digitalWrite(relayPin, HIGH); // Turn on the relay

} else {

digitalWrite(relayPin, LOW); // Turn off the relay

}

// Send data to serial

Serial.print("Temperature: ");

Serial.print(temperature);

Serial.println(" °C");

delay(1000); // Adjust delay as needed

}

=====================================================================================

**SETUP python**

Goto: “<https://www.python.org/downloads/>”

1. **Step 1: Install Python**
2. **Download Python:** Go to the official Python website at [python.org](https://www.python.org/downloads/) and download the latest version of Python for your operating system (Windows, macOS, or Linux).
3. **Run the Installer:** Run the downloaded installer and make sure to check the box that says "Add Python to PATH" during installation.
4. **Verify Installation:** Open a command prompt or terminal window and type the following command to check if Python was installed successfully:
5. bashCopy code
6. python --version
7. You should see the Python version number.
8. **Step 2: Install pyserial and requests**
9. **Install pyserial:** Open a command prompt or terminal window and type the following command:
10. bashCopy code
11. pip install pyserial
12. **Install requests:** Type the following command:
13. bashCopy code
14. pip install requests
15. **Step 3: Example Code for Reading Data from Arduino Serial**
16. Now, let's create a simple Python script to read data from Arduino serial. Suppose your Arduino is sending temperature data over the serial port. Connect your Arduino, find the correct port (COMx for Windows, /dev/ttyUSBx or /dev/ttyACMx for Linux), and replace **'YOUR\_ARDUINO\_PORT'** in the code below with the correct port.

.

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**PYTHON CODE:**

import serial

import requests

import time

# Arduino serial port, update it based on your setup

arduino\_port = 'COM3' # Change this to your Arduino's port

baud\_rate = 9600

# ThingSpeak API key and channel ID

api\_key = 'YOUR\_THINGSPEAK\_API\_KEY'

channel\_id = 'YOUR\_THINGSPEAK\_CHANNEL\_ID'

# Function to read temperature from Arduino via serial

def read\_temperature(serial\_port):

try:

# Read data from Arduino

data = serial\_port.readline().decode('utf-8').strip()

temperature = float(data)

return temperature

except Exception as e:

print(f"Error reading temperature: {e}")

return None

# Function to upload data to ThingSpeak

def upload\_to\_thingspeak(api\_key, channel\_id, temperature):

try:

url = f'https://api.thingspeak.com/update?api\_key={api\_key}&field1={temperature}'

response = requests.get(url)

if response.status\_code == 200:

print(f"Data uploaded to ThingSpeak: {temperature}")

else:

print(f"Failed to upload data to ThingSpeak. Status code: {response.status\_code}")

except Exception as e:

print(f"Error uploading data to ThingSpeak: {e}")

# Main program

if \_\_name\_\_ == "\_\_main\_\_":

try:

# Open serial connection to Arduino

arduino\_serial = serial.Serial(arduino\_port, baud\_rate, timeout=1)

while True:

# Read temperature from Arduino

temperature = read\_temperature(arduino\_serial)

if temperature is not None:

# Upload data to ThingSpeak

upload\_to\_thingspeak(api\_key, channel\_id, temperature)

# Delay for a while before the next reading

time.sleep(15)

except KeyboardInterrupt:

# Close the serial connection when the program is interrupted

arduino\_serial.close()

print("Serial connection closed.")

except Exception as e:

print(f"An error occurred: {e}") =====================================================================================**Note:**

* Make sure to install the necessary Python libraries (**requests**) using **pip install requests**.
* Replace **'COMX'** with the appropriate serial port your Arduino is connected to.
* Adjust the relay control logic and temperature threshold based on your specific requirements.
* Customize the dashboard URL and payload format according to your IT dashboard requirements.

Manual logging of data:

1. Open the Arduino IDE and upload your Arduino code to the board.
2. Open the Serial Monitor (you can find it in the Tools menu).
3. Once the Serial Monitor is open, you should start seeing the temperature values being printed there.

Now, to copy the data:

1. Select the temperature values that you want to copy. Click and drag the mouse over the numbers in the Serial Monitor to highlight them.
2. Right-click on the highlighted values and select "Copy" from the context menu, or use the keyboard shortcut (Ctrl+C on Windows or Command+C on Mac).

Now, to paste the data into a CSV file:

1. Open a text editor or a spreadsheet program (like Microsoft Excel or Google Sheets).
2. In the text editor or spreadsheet, position the cursor where you want to paste the data.
3. Right-click and select "Paste" from the context menu, or use the keyboard shortcut (Ctrl+V on Windows or Command+V on Mac).
4. Save the document as a CSV file. In many spreadsheet programs, you can use the "Save As" option and choose the CSV format.

Now, you should have a CSV file with your temperature data. Remember that this method is manual and might not be suitable for continuous or automated logging. For automated logging, you would need to write a script (in Python, for example) to read the serial data and write it to a CSV file. If you are interested in automating the process, let me know, and I can provide guidance on that as well.

**Getting started with Things Speak**

**Step 1: login to** [**https://thingspeak.com/**](https://thingspeak.com/)