WEATHER STATION USING RASPBERRY PI PICO AND SENSORS

Vijay V
Hari Prasath S
Gokulnath S

INTRODUCTION

In the era of increasing environmental awareness and the Internet of Things (IoT), the Raspberry Pi Pico Weather Station project offers a cost-effective and accessible solution for weather data collection and monitoring. This project aims to design and implement a weather station using the Raspberry Pi Pico microcontroller, which is compact, energy-efficient, and suitable for a wide range of IoT applications.

The Raspberry Pi Pico Weather Station project incorporates various sensors to measure key meteorological parameters, such as temperature, humidity, pressure, and light intensity. The collected data is processed and transmitted to a cloud-based server for real-time monitoring and analysis. This project emphasizes user-friendliness, scalability, and adaptability to different environmental monitoring needs.

The project utilizes Python programming for the Raspberry Pi Pico, ensuring ease of development and customization. Additionally, it employs wireless communication protocols, such as Wi-Fi or LoRa, for data transmission, enabling remote access to weather data through web-based dashboards or mobile applications.

EXISTING SYSTEM

Forecasting was mostly based on weather pattern observation. Over the years, the study of weather patterns has resulted in various techniques for rainfall forecasting. Present rainfall forecasting embodies a combination of computer models, interpretation, and an acquaintance of weather patterns. The following technique was used for existing weather prediction.

- Use of a barometer
- Looking at the sky
- Nowcasting
- Analog technique
- Numerical Weather Prediction model
- Radar
- Weather satellites

PROPOSED SYSTEM

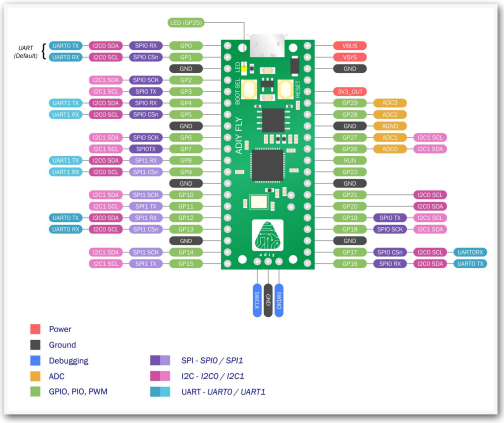
This proposal outlines the development of a weather forecasting system that leverages the capabilities of the Raspberry Pi microcontroller in conjunction with environmental sensors, specifically the DHT11 for temperature and humidity measurements and a rain sensor for precipitation detection. The proposed system aims to enhance local weather predictions and provide valuable data for various applications, including agriculture, urban planning, and disaster preparedness.

Key Components and Functionality

- **Raspberry Pi**: The Raspberry Pi serves as the central processing unit for the system. Its computational power and connectivity options make it an ideal platform for data collection, analysis, and transmission.
- **DHT11 Sensor**: The DHT11 sensor will be used to measure temperature and humidity levels in the environment. These measurements are essential for assessing atmospheric conditions, which are fundamental for weather forecasting.
- **Rain Sensor**: The rain sensor is employed to detect precipitation and rainfall intensity. The collected data can assist in predicting weather patterns and possible flooding events.
- **Data Collection**: The Raspberry Pi continuously collects data from the DHT11 and rain sensor. The data collected includes temperature, humidity, and rain intensity mostly based on weather pattern observation
- **Data Sharing**: The system can transmit weather data to a central server or cloud platform, making it accessible to a broader audience and enhancing collaborative weather forecasting efforts
- **Notifications**: The system may be programmed to send alerts and notifications to users based on changing weather conditions, ensuring public safety and aiding in decision-making.

The proposed system provides a cost-effective and accessible solution for local weather forecasting. It utilizes off-the-shelf components and open-source software, making it suitable for educational purposes and community projects. This project aims to contribute to improved weather predictions at the local level and foster a greater understanding of the potential applications of microcontrollers and sensors in environmental monitoring and forecasting.

BLOCK DIAGRAM



RASPBERRY PI PICO ARCHITECTURE

CONNECTIONS:

Connections between the DHT-11 sensor module and RPI pico.

DHT-11	PICO
VCC (+)	3.3V
OUT	GP15
GND (-)	GND

Connections between the RainSensor module and RPI pico.

RAIN SENSOR	PICO
VCC (+)	3.3V
GND(-)	GND
AO	AGND
DO	GP26

EXPLANATION

The hardware setup and Mapping guide for the DHT11 Sensor with Raspberry Pi Pico board & the rain sensor is done accordingly from the above mentioned tabulation precisely.

After the mapping is done the software setup is begun, firstly the raspberry pi pico is connected to the computer with the help of micro USB to USB cable.In this project the raspberry pi pico is interfaced using the micro python so the Thonny text editor is opened and the raspberry pi pico is detected and the desired ".uf2" file extension for the raspberry pi pico board is installed only then the microcontroller can be programmed according to our need.

Once all these procedures have been completed the scripting process is begun the micro python libraries for the DHT-11and RainSensors is imported, only after that the coding process is initiated since the sensors would not work without it's existing libraries from which the functions that we are going to use

After the completion of the coding the primary script is named as "**main.py**" because in micro python the file which has to be automatically executed is named as main followed by an extension of a python file that is ".py"

After the Upload code, You will see that the Pico board Start displaying the weather temperature , humidity and the rainfall percentage in the output part of the thonny IDE

LIBRARY & CODE

DHT-11 library

```
import array
import micropython
import utime
from machine import Pin
from micropython import const

class InvalidChecksum(Exception):
    pass

class InvalidPulseCount(Exception):
    pass

MAX_UNCHANGED = const(100)
MIN_INTERVAL_US = const(200000)
HIGH_LEVEL = const(50)
EXPECTED_PULSES = const(84)

class DHT11:
    _temperature: float
    _humidity: float

    def __init__(self, pin):
```

```
        self._pin = pin
        self._last_measure = utime.ticks_us()
        self._temperature = -1
        self._humidity = -1

    def measure(self):
        current_ticks = utime.ticks_us()
        if utime.ticks_diff(current_ticks, self._last_measure) < MIN_INTERVAL_US and (
            self._temperature > -1 or self._humidity > -1
        ):
            # Less than a second since last read, which is too soon according
            # to the datasheet
            return

        self._send_init_signal()
        pulses = self._capture_pulses()
        buffer = self._convert_pulses_to_buffer(pulses)
        self._verify_checksum(buffer)

        self._humidity = buffer[0] + buffer[1] / 10
        self._temperature = buffer[2] + buffer[3] / 10
        self._last_measure = utime.ticks_us()

    @property
    def humidity(self):
        self.measure()
        return self._humidity

    @property
    def temperature(self):
        self.measure()
        return self._temperature

    def _send_init_signal(self):
        self._pin.init(Pin.OUT, Pin.PULL_DOWN)
        self._pin.value(1)
        utime.sleep_ms(50)
        self._pin.value(0)
        utime.sleep_ms(18)

    @micropython.native
    def _capture_pulses(self):
        pin = self._pin
        pin.init(Pin.IN, Pin.PULL_UP)

        val = 1
        idx = 0
        transitions = bytearray(EXPECTED_PULSES)
        unchanged = 0
        timestamp = utime.ticks_us()

        while unchanged < MAX_UNCHANGED:
            if val != pin.value():
                if idx >= EXPECTED_PULSES:
                    raise InvalidPulseCount(
                        "Got more than {} pulses".format(EXPECTED_PULSES)
                    )
                now = utime.ticks_us()
                transitions[idx] = now - timestamp
                timestamp = now
                idx += 1

                val = 1 - val
```

```
                unchanged = 0
            else:
                unchanged += 1
        pin.init(Pin.OUT, Pin.PULL_DOWN)
        if idx != EXPECTED_PULSES:
            raise InvalidPulseCount(
                "Expected {} but got {} pulses".format(EXPECTED_PULSES, idx)
            )
        return transitions[4:]

    def _convert_pulses_to_buffer(self, pulses):
        """Convert a list of 80 pulses into a 5 byte buffer
        The resulting 5 bytes in the buffer will be:
            0: Integral relative humidity data
            1: Decimal relative humidity data
            2: Integral temperature data
            3: Decimal temperature data
            4: Checksum
        """
        # Convert the pulses to 40 bits
        binary = 0
        for idx in range(0, len(pulses), 2):
            binary = binary << 1 | int(pulses[idx] > HIGH_LEVEL)

        # Split into 5 bytes
        buffer = array.array("B")
        for shift in range(4, -1, -1):
            buffer.append(binary >> shift * 8 & 0xFF)
        return buffer

    def _verify_checksum(self, buffer):
        # Calculate checksum
        checksum = 0
        for buf in buffer[0:4]:
            checksum += buf
        if checksum & 0xFF != buffer[4]:
            raise InvalidChecksum()
```

Main.py code

```
from machine import Pin
import utime as time
import utime
from dht import DHT11, InvalidChecksum

adc = machine.ADC(26)
conversion_factor = 100 / (65535)

while True:
    time.sleep(1)
    pin = Pin(15, Pin.OUT, Pin.PULL_DOWN)
    sensor = DHT11(pin)
    t  = (sensor.temperature)
    h = (sensor.humidity)
    print("READINGS FROM DHT-11 SENSOR :")
    print("Temperature: {}".format(sensor.temperature))
    print("Humidity: {}".format(sensor.humidity))

    rainCoverage = 100 - (adc.read_u16() * conversion_factor)
    print("READING FROM THE RAIN SENSOR : ")

    print(round(rainCoverage, 1), "%")
    utime.sleep_ms(1000)
```

ADVANTAGES AND APPLICATIONS

Advantages:

- **Affordability**: Raspberry Pi Pico, DHT11, and rain sensors are cost-effective components, making the weather station accessible to hobbyists, educational institutions, and communities with limited budgets.
- **Customizability**: The system can be tailored to specific needs and expanded with additional sensors for a more comprehensive environmental monitoring setup.
- **Data Accuracy**: The DHT11 sensor provides accurate temperature and humidity measurements, while the rain sensor detects precipitation, contributing to reliable data collection.
- **Real-Time Monitoring** : The weather station can provide real-time weather data, allowing users to make timely decisions based on current conditions.
- **Data Storage** : Data can be stored locally on the Raspberry Pi Pico, allowing for historical data analysis and reference.
- **Data Transmission** : The system can transmit weather data to remote servers or cloud platforms, enabling collaboration, analysis, and long-term data storage.

Applications:

- **Agriculture**: Farmers can use this weather station to monitor temperature, humidity, and rainfall, aiding in crop management, irrigation, and pest control decisions.
- **Gardening**: Home gardeners can use the weather station to optimize watering schedules, protect plants from extreme conditions, and create a healthier garden environment.
- **Urban Planning**: City planners can gather local weather data for infrastructure development, flood prevention, and city design.
- **Weather Forecasting**: By sharing data with meteorological agencies, communities can contribute to local weather forecasting and early warning systems.
- **Environmental Research** : Scientists and researchers can use the weather station to gather data for environmental studies, climate change research, and ecological monitoring.

- **Education** : The weather station serves as an educational tool for schools and universities, allowing students to learn about weather patterns, sensor technology, and data analysis.
- **Disaster Preparedness** : Communities can use the system to monitor extreme weather events, such as heavy rainfall or storms, and prepare accordingly.
- **Home Automation** : Weather data can be used to automate home systems like HVAC and irrigation, increasing energy efficiency and cost savings.
- **Smart Agriculture** : The data collected can be used in precision agriculture to optimize farming practices, such as planting and harvesting times.

REFERENCES

1. https://gargicollege.in/wp-content/uploads/2020/03/weather_forecast.pdf
2. https://iotdesignpro.com/articles/build-your-own-weather-station-using-raspberry-pi-pico-w
3. https://diyprojectslab.com/raspberry-pi-pico-weather-station-dht11/#:~:text=Schematics for Raspberry Pi Pico Weather Station&text=Connect the DHT11 digital output,Pi Pico GND(-) Pin.
4. https://github.com/panchalnikunj/Raspberry-Pi-Pico-Projects/tree/main/Weather Station Using DHT11 and Raspberry Pi Pico

CONCLUSION

In summary, a weather station based on Raspberry Pi Pico, DHT11, and a rain sensor is a versatile tool with a wide range of practical applications, making it suitable for various user groups, from individuals and hobbyists to professionals and researchers. Its affordability, ease of customization, and data accuracy make it a valuable addition to environmental monitoring and decision-making processes.