Introduction

In this lab, we will discuss the basic understanding about the communication protocols.

Sensors

Sensors are electronic components that detect changes in the environment and convert them into an electrical signal. Sensor are electrical, opto-electrical, or electronic devices composed of specialty electronics or otherwise sensitive materials, for determining if there is a presence of a particular entity or function. They are used in a wide range of applications, from consumer electronics to industrial machinery, and are an essential component of many modern systems, including Internet of Things (IoT) devices, autonomous vehicles, and smart home systems.

There are many different types of sensors, each designed to detect different types of changes in the environment. Some common examples include:

Temperature sensors: detect changes in temperature and are commonly used in thermostats and weather monitoring systems.

Pressure sensors: detect changes in pressure and are used in applications such as monitoring tire pressure or controlling industrial machinery.

Motion sensors: detect motion and are commonly used in security systems, automatic doors, and gaming consoles.

Light sensors: detect changes in light levels and are used in applications such as cameras, automatic lighting systems, and display screens.

Proximity sensors: detect the presence of nearby objects and are commonly used in smartphones and robotic systems.

Accelerometers: detect changes in acceleration and are used in applications such as gaming controllers, fitness trackers, and smartphones.

Sensors play a critical role in many modern systems, enabling them to detect and respond to changes in the environment and providing valuable data for monitoring and control purposes. Calibration is an important step in ensuring the accuracy and reliability of sensor measurements, particularly in applications where precise and consistent measurements are critical, such as in medical devices, aerospace, and industrial monitoring systems. Regular

calibration helps to maintain the sensor's accuracy and reliability over time, improving the quality and consistency of the data collected by the sensor.

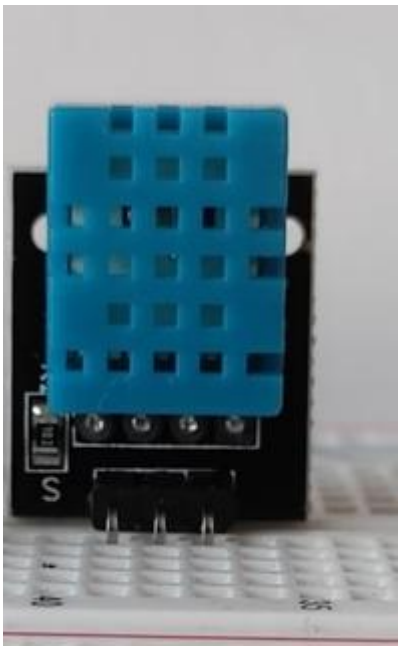
Sensors are generally classified into two major types:

i. **Contactless sensors**

ii. **Contacting sensors**

DHT11 Sensor: Measuring temperature and humidity

The DHT11 sensor is a digital temperature and humidity sensor that is commonly used for monitoring environmental conditions in various applications. It is a low-cost sensor module that provides reliable temperature and humidity readings with relatively simple interfacing requirements.



Some technical characteristics of the DHT11

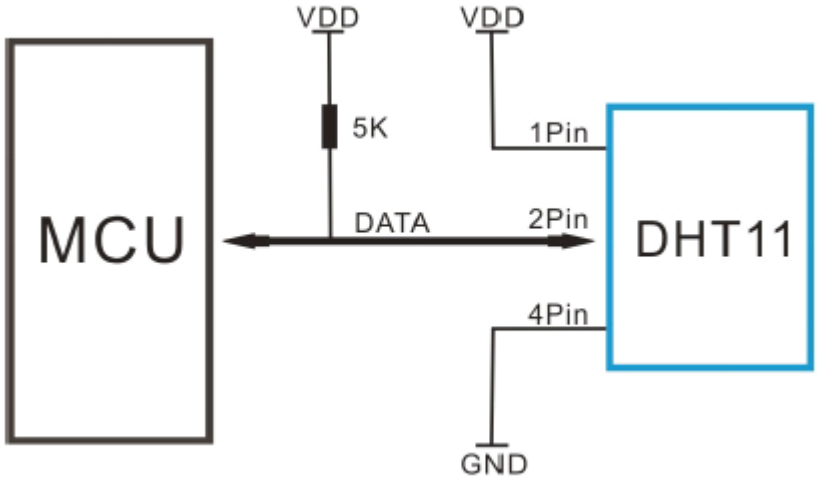
DHT11 Specifications

Features	DHT11
Temperature accuracy	± 2°C
Humidity accuracy	± 5%
Temperature range	0-50 °C
Humidity range	20-90%
Sampling	1/s
Supply voltage	3-5.5V
Current consumption	~2.5mA

The numbering is done from the left when you hold the sensor facing you (the part with the grid in front of you).

DHT11 Module	ESP32-C6
1 (S)	GPIO-PIN
2	3V3-5V
3 (-)	GND

These sensors contain a chip that does analog to digital conversion and spit out a digital signal with the temperature and humidity. This makes them very easy to use with any microcontroller.



To import the DHT11 sensor libraries into your project, follow these steps:

Download Libraries: Ensure you have the necessary libraries for the DHT11 sensor. These typically include a C source file (dht11.c) and a header file (dht11.h). You can obtain these

libraries from reliable sources such as the manufacturer's website or trusted repositories like GitHub.

Create Components Folder: Within your project directory, create a folder named components if it doesn't already exist. This folder will hold all the external components' libraries and source files.

Place Libraries in Components Folder: Move the dht11.c and dht11.h files into the components folder you created in the previous step.

Include Libraries in Your Project: In your project source code files where you intend to use the DHT11 sensor functionalities, include the dht11.h header file.

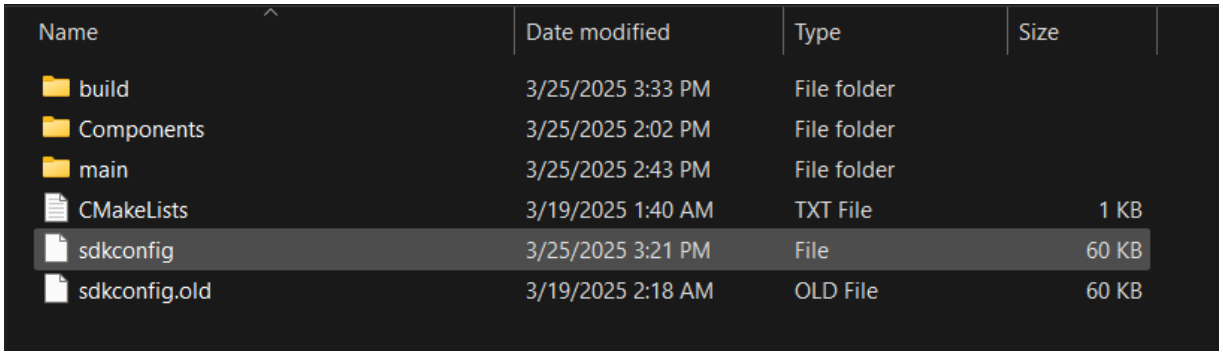
Compile and Link: Ensure that your build system is configured to compile and link the dht11.c source file along with your project files. (**CMakeLists.txt** enables you to specify dependencies on external libraries and components, including the DHT11 sensor library in this case. This makes it easier to manage dependencies and ensures that your project can be built consistently across different environments.) import the DHT11 sensor libraries into your project, follow these steps:

Design Problem:

Design a system to measure temperature using the DHT11 sensor, providing accurate and reliable temperature readings.

Steps to be Followed

i. Set Up ESP-IDF Project



ii. Include Required Libraries:

my_new_project.c > ...

```c
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_system.h"
#include "driver/gpio.h"
#include "sdkconfig.h"
#include "esp_log.h"
#include "dht11.h"
```

iii. Define Configuration Constants

CMakeLists.txt

```
idf_component_register(SRCS "dht11.c" "my_new_project.c"
                    INCLUDE_DIRS ".")
```

```c
#define DHT_GPIO GPIO_NUM_4  // Change to the GPIO pin you are using
```

iv. Implement dht_test Function

```c
void dht_test(void *pvParameter) {
    ESP_LOGI("DHT", "Starting DHT11 Sensor Readings");

    while (1) {
        int16_t temperature = 0, humidity = 0;
        esp_err_t result = dht_read_data(DHT_TYPE_DHT11, DHT_GPIO,
&humidity, &temperature);

        if (result == ESP_OK) {
            printf("Temperature: %d.%d°C\n", temperature / 10,
temperature % 10);
            printf("Humidity: %d.%d%%\n", humidity / 10, humidity % 10);
        } else {
            ESP_LOGE("DHT", "Failed to read sensor data");
        }

        vTaskDelay(pdMS_TO_TICKS(2000));  // Delay 2 seconds before the
next read
    }
}
```

v. Implement app_main Function

```
void app_main(void) {
    xTaskCreate(&dht_test, "dht_test", 2048, NULL, 5, NULL);
}
```

vi. Compile and Flash

```
Bootloader binary size 0x6880 bytes. 0x780 bytes (7%) free.
[6/7] Generating binary image from built executable
esptool.py v4.8.1
Creating esp32 image...
Merged 2 ELF sections
Successfully created esp32 image.
Generated C:/Espressif/frameworks/esp-idf-v5.3.1/my_new_project/build
[7/7] cmd.exe /C "cd /D C:\Espressif\frameworks\esp-idf-v5...ks/esp-i
my_new_project.bin binary size 0x2c4b0 bytes. Smallest app partition
```

vii. Monitor Output

```
I (297) main_task: Returned from app_main()
Temperature: 25.0°C, Humidity: 58.0%
Temperature: 25.0°C, Humidity: 58.0%
Temperature: 25.0°C, Humidity: 58.0%
Temperature: 25.0°C, Humidity: 58.0%
Temperature: 25.0°C, Humidity: 58.0%
```

Implement the dht_test function as provided. This function reads temperature and humidity data from the DHT11 sensor. Ensure to include the necessary setup code such as enabling pull-up resistors if required (based on CONFIG_EXAMPLE_INTERNAL_PULLUP). Initialize DHT11 sensor by gpio_set_direction Use the dht_read_float_data function to read data from the DHT11 sensor.Print the temperature and humidity readings to the console.Add a delay using vTaskDelay to control the frequency of sensor readings

Expected Output

```
Temperature: 25.0°C, Humidity: 58.0%
I (297) main_task: Returned from app_main()
Temperature: 25.0°C, Humidity: 58.0%
Temperature: 25.0°C, Humidity: 58.0%
Temperature: 25.0°C, Humidity: 58.0%
Temperature: 25.0°C, Humidity: 58.0%
Temperature: 25.0°C, Humidity: 57.0%
Temperature: 25.0°C, Humidity: 57.0%
Temperature: 25.0°C, Humidity: 57.0%
Temperature: 25.0°C, Humidity: 57.0%
Temperature: 25.0°C, Humidity: 57.0%
```

Conclusion

In this lab we understand how to use sensors using Esp 32 and what are the applications of ESP-32 in the use of sensors. We also understand how different libraries are required to make the esp32 compatible with sensors and we learn how we can establish an application using DHT sensors and ESP-32.

Github-repo Link:

https://github.com/SHAHEER-DASTGIR/DHT-11-Temperature-Sensor.git