DHT22 Temperature and Humidity Sensor

by Reece Landry, Alex Biscoveanu, Jachak Sekhon
Last update: April 10, 2025

Guide has been tested on

BeagleY-AI (Target):	**Debian 12.2.0**
VM OS (host):	**Debian 12.2.0**

This document guides the user through

1. Connecting the DHT22 to the BeagleY-AI
2. Sending and receive commands via C++ code
3. Decoding received data in C++

Table of Contents

Formatting

1. Commands for the host Linux's console are shown as:
   (host) $ echo "Hello PC world!"
2. Commands for the target (BeagleY-AI) Linux's console are shown as:
   (byai) $ echo "Hello embedded world!"
3. Almost all commands are case sensitive

Revision History

- April 10, 2025

1. DHT Pinout

The DHT22 has 4 pins as shown below.

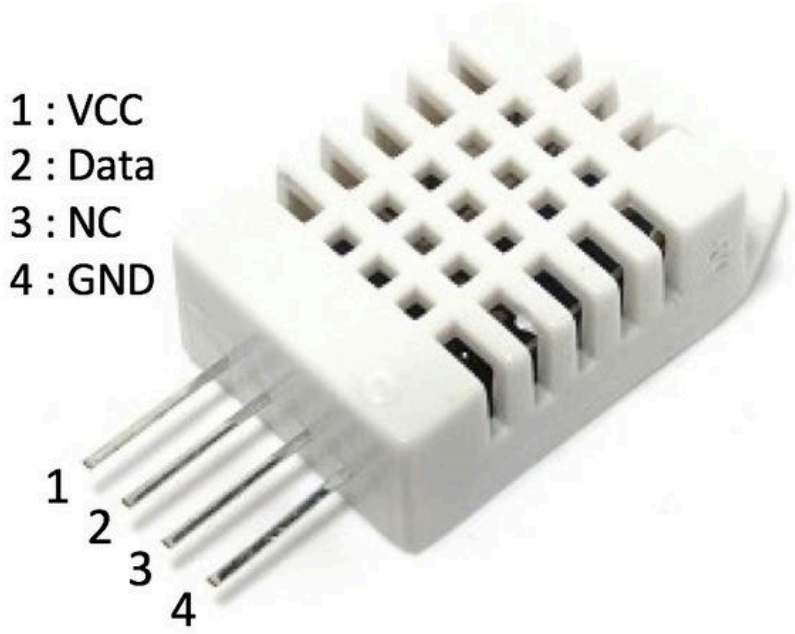

Figure 1: DHT22 Pinout.

The VCC pin can handle 3.3-5.5V because the DHT22 has an internal level shifter. Also, since there is only one data pin, it uses 1-Wire communication for all data in and out.

2. Connecting the DHT22 to the BeagleY-AI

The DHT22 needs a pull up resistor in order to pull high when the device is in Sleep mode. We will also need to leverage the Real-Time processing of the R5 so we need to ensure to use a pin that the R5 has access to. We are going to use GPIO23 for this guide. We will need 2 10kΩ resistors to put in parallel to create a combined 5kΩ resistance. These resistors will go between the power and the data lines. Once everything has been connected as shown in the image below, it is time to start programming. Make sure to configure the pin on each boot!

```
(byai) $ gpioset gpiochip0 7=1
```

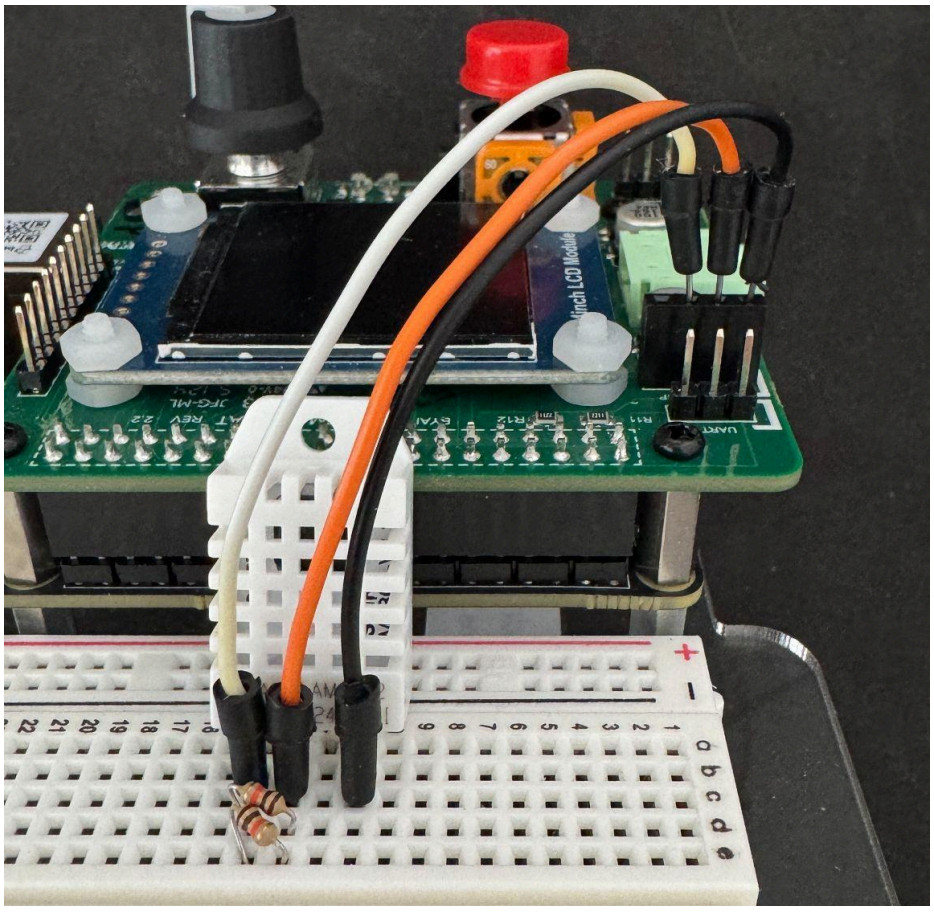

Figure 2: DHT22 connection to BeagleY-AI.

3. DHT22 Controlled Via C++

3.1 Request Data

Before reading or sending any data, it is important to give the DHT22 at least 2 seconds to power on and self calibrate.

```
k_busy_wait(DhtData::SLEEP_2_AND_HALF_S);
```

Now that we are sure the device is one and ready, we can send a command to wake the device up. In order to set the device to High Speed mode, we must pull the data line LOW for at least 1ms and then set it back to HIGH for 30µs- before configuring the pin for input.

```
gpio_pin_configure_dt(&mDhtPin, GPIO_OUTPUT_ACTIVE);
gpio_pin_set_dt(&mDhtPin, 0);
delayNS(DhtData::SLEEP_2MS);  // Hold LOW for 2 ms
gpio_pin_set_dt(&mDhtPin, 1);
delayNS(DhtData::SLEEP_30US); // 30 µs HIGH
gpio_pin_configure_dt(&mDhtPin, GPIO_INPUT);
```

```
auto DHT::delayNS(uint32_t ns) -> void {
   uint32_t target = k_cycle_get_32() + k_ns_to_cyc_near32(ns);
   while (k_cycle_get_32() < target);
}
```

This tells the DHT22 that we want to receive a reading. Note you can only call for a new reading at most once per 2 seconds. Now that we have told the sensor to wake up, we must confirm it has in fact woken up. We can do this by checking the input for 80µs of LOW reading followed by 80µs of HIGH reading.

```
// Wait for sensor to pull LOW
int timeout = 0;
while (gpio_pin_get_dt(&mDhtPin) == 1) {
   if (++timeout > 100) { // Added a little extra 20µs of padding
      break;
   }
   delayNS(DhtData::SLEEP_1MS);
}
// Wait for sensor to pull HIGH
timeout = 0;
   while (gpio_pin_get_dt(&mDhtPin) == 0) {
      if (++timeout > 100) { // Added a little extra 20µs of padding
         break;
      }
   delayNS(DhtData::SLEEP_1MS);
}
```

3.2 Read Data

Reading data back from the DHT22 is the most time sensitive part of the communication. The sensor is going to return data in this format:

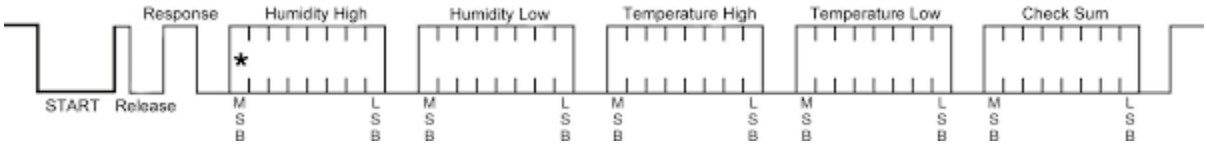

Figure 3: DHT22 byte sequence order.

The sensor is going to output 40 bits, high bit first. The outputted data is Humidity High, Humidity Low, Temperature High, Temperature Low, and a Parity bit for checksum. To start, read the first 40 bits by listening for a LOW signal for 50ms then time how long the line goes HIGH for.

Bit data “0”: LOW for 50ms and HIGH for 26-28ms.

Bit data “1”: LOW for 50ms and HIGH for 70ms.

```
// Read 40 bits from the sensor
uint8_t data[5] = {0};
for (int i = 0; i < 40; ++i) {
   // Wait for LOW
   uint32_t timeout = 0;
   while (gpio_pin_get_dt(&mDhtPin) == 0) {
      if (++timeout > 100) break;
         delayNS(DhtData::SLEEP_1MS);
      }
   }

   // Measure HIGH duration
   timeout = 0;
   while (gpio_pin_get_dt(&mDhtPin) == 1) {
      if (++timeout > 150) break;
         delayNS(DhtData::SLEEP_1MS);
      }
   }

   // Determine “1” or “0”
   uint8_t bit = (timeout > 40) ? 1 : 0;

   // Store bit in data
   if (bit) {
      data[(i / 8)] |= (1 << (7 - (i % 8)));
   }
}
```

Once communication is complete, the sensor will go back into Sleep mode awaiting another wake up signal.

3.3 Decode Data

Now that we have read data, we must decode it to figure out what the humidity and temperature is. The first thing we want to check is the checksum to make sure we have a valid reading.

```
uint8_t checksum = data[0] + data[1] + data[2] + data[3];
if (checksum == data[4]) {
   // Checksum is valid!
}
```

Humidity is the first 16 bits, high bit first. Temperature is the next 16 bits, high bit first. The first bit (MSB) is “1” if the temperature is negative and “0” when the temperature is positive. The remaining 15 bits are the temperature reading. Both temperature and humidity are 10 times higher than the actual value.

```
auto DHT::decodeDHT22Temperature(uint8_t msb, uint8_t lsb) -> float {
   uint16_t raw = (static_cast<uint16_t>(msb) << 8) | lsb;
   bool negative = raw & 0x8000;
   int16_t value = raw & 0x7FFF;
   if (negative) value = -value;
   return value / 10.0f;
}

auto DHT::decodeDHT22Humidity(uint8_t msb, uint8_t lsb) -> float {
   uint16_t raw = (static_cast<uint16_t>(msb) << 8) | lsb;
   return raw / 10.0f;
}
```

That's it! You know have an accurate temperature and humidity reading. Don’t forget to sleep for at least 2 seconds before sampling again.

```
// Maintain 2.5s sampling interval
k_busy_wait(DhtData::SLEEP_2_AND_HALF_S);
```

4. Troubleshooting

Common issues:

- If you are only reading HIGH values, it is likely you have not properly sent the wakeup command. Make sure you have configured the line for output and are pulling low for at least 1ms.
- If the numbers you decoded are coming out wrong, you may have an alignment issue. Try reading the first 41 bits instead of 40 and discard the first bit read.
- If your code seems to start and run but not work, ensure you are using k_busy_wait() for your delays rather than k_msleep()

Tips:

- After trying to wake up your DHT22 read for an acknowledgement and save whether you got a good ack or not to shared memory before trying to decode the signal.