# Arduino UNO R4 Wi-Fi CAN BUS Project

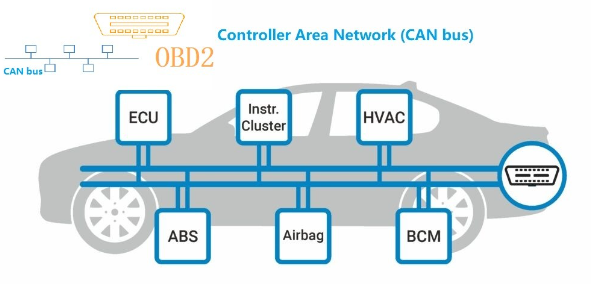
Difficult Level: 

## Principle

First of all, let us know about CAN BUS.

**CAN bus,** short for **Controller Area Network bus**, is a robust and widely used communication protocol primarily employed in automotive and industrial applications. It was originally developed by Bosch in the 1980s to address the increasing complexity of automotive electronic systems.

CAN bus is a multi-master, message-oriented protocol designed for high-speed, reliable communication between electronic control units (ECUs) in vehicles or industrial systems. It uses a differential serial bus to transmit data between nodes without the need for a centralized controller. One of the key features of CAN bus is its reliability and robustness. It employs error detection and error handling mechanisms, including cyclic redundancy checks (CRCs), to ensure data integrity even in noisy environments. It supports a scalable network architecture, allowing for easy expansion by adding more nodes without significant changes to the overall system. This makes it suitable for applications ranging from simple vehicle subsystems to complex automotive or industrial networks It offers deterministic real-time communication, making it suitable for applications where timing constraints are critical, such as engine control, transmission systems, anti-lock braking systems (ABS), and electronic stability control (ESC) in vehicles. And it is a cost-effective solution for communication within vehicles and industrial machinery compared to alternatives like Ethernet or USB. Its simplicity and widespread adoption contribute to lower implementation costs.



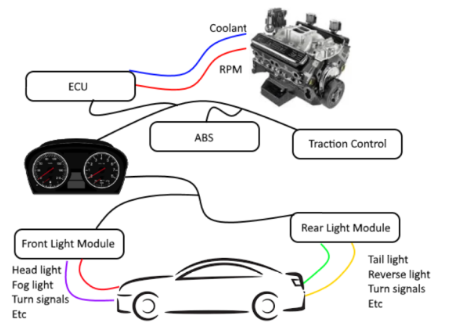
## Typical Scenarios

**Automotive:** CAN bus is extensively used in automotive applications for communication between various ECUs, including engine control units (ECUs), transmission control units (TCUs), airbag control modules, ABS systems, instrument clusters, and more. It enables data exchange for vehicle diagnostics, engine management, safety systems, and infotainment.

**Industrial Automation:** In industrial automation, CAN bus facilitates communication between programmable logic controllers (PLCs), sensors, actuators, human-machine interfaces (HMIs), and other devices on the factory floor. It enables real-time control and monitoring of manufacturing processes, machinery, and industrial robots.

**Agricultural and Heavy Machinery:** CAN bus is also used in agricultural machinery, construction equipment, and other heavy-duty vehicles for tasks such as precision farming, equipment control, and fleet management.

Aviation and Aerospace: CAN bus finds applications in aerospace systems for communication between avionics components, flight control systems, navigation systems, and on-board computers.



## Benefits

**What is the benefit of combining Arduino Uno R4 Wi-Fi board with CAN BUS?**

Combining an Arduino Uno R4 Wi-Fi board with a CAN bus interface provides several features and capabilities, especially in the context of embedded systems and automotive applications

Here are some key features and benefits of this combination:

* **Wireless Connectivity:** The Arduino Uno R4 Wi-Fi board allows wireless communication, enabling remote monitoring and control of devices. This feature can be leveraged for IoT applications where connectivity to the internet or local network is required.
* **CAN Bus Interface:** The CAN (Controller Area Network) bus is a robust and widely used communication protocol in automotive and industrial applications. It allows for reliable communication between microcontrollers and various devices such as sensors, actuators, and other control modules.
* **Integration of CAN Communication with Wireless Control:** By combining Wi-Fi capability with CAN bus communication, you can create systems that not only interact with local CAN bus networks but also remotely access and control them over a wireless network. This can be useful for applications such as vehicle diagnostics, fleet management, and remote monitoring/control of industrial machinery.
* **Data Logging and Analysis**: With the Arduino Uno R4 Wi-Fi board's capability for wireless connectivity and the CAN bus interface, you can collect data from CAN bus networks and log it to remote servers or cloud platforms for further analysis. This data can then be used for performance monitoring, predictive maintenance, or optimization of systems.
* **Remote Firmware Updates:** The wireless connectivity of the Arduino Uno R4 Wi-Fi board enables remote firmware updates, allowing you to deploy software patches or new features to devices connected to the CAN bus without physically accessing them. This feature is valuable for maintaining and updating large fleets of vehicles or distributed industrial systems.
* **Integration with Existing Infrastructure:** Many existing automotive and industrial systems utilize the CAN bus for communication. By integrating a Wi-Fi-enabled Arduino Uno R4 board with CAN bus capability, you can extend the functionality of these systems by adding wireless connectivity without the need for significant hardware modifications.
* **Flexible Development Platform:** Arduino boards are known for their ease of use and extensive community support. By choosing the Arduino Uno R4 Wi-Fi board with CAN bus support, you get access to a vast ecosystem of libraries, tutorials, and resources, making it easier to develop and prototype applications that leverage both wireless communication and CAN bus technology.

By now, we have understood the characteristics and usage scenarios of CAN BUS through the content above. Let's imagine a scenario: If you prefer to place your sensors outdoors rather than indoors, and you wish to collect data through these outdoor sensors, transmitting the data back indoors via an Arduino using the CAN BUS, and then another Arduino device indoors not only aggregates this collected data but also integrates it with our previously set up Home Assistant for centralized data display. This would definitely be a very cool thing to do. You might wonder, why not use Wi-Fi or other wireless methods for transmission? Why use the CAN bus protocol instead? Here's where the characteristics of CAN bus protocol transmission come into play. Only two arbitrary wires are needed to establish connection and data transmission between two CAN modules, which is very convenient. If there are multiple devices, they can simply be connected in series to the bus line to collect data from any number of devices, significantly reducing wiring costs. Deploying Wi-Fi or other wireless devices would incur higher deployment costs and may not necessarily be as stable. Therefore, CAN bus is certainly more cost-effective!

Let’s get start our project!

## Project Details:

Now let's build a project. I need to use an Arduino UNO R4 Wi-Fi to connect a temperature and humidity sensor (DHT11) and collect temperature and humidity data. This data will be transmitted to another Arduino with a CAN module using MQTT protocol. The data is then sent to the Home Assistant data center.

### Hardware Preparation

* 2 x Arduino UNO R4 board (Wi-Fi or Minima)
* 2 x CAN Bus module
* 2 pair twist wire (you can just use two copper wire or network cable’s wire)
* 2 x USB-C programming cable
* 1 x 52Pi experiment platform [Optional]
* 1 x Temperature & Humidity sensor or BMP280 air pressure meter
* 20 x Male to Female Jumper wire

### Software Preparation

* Arduino IDE
* Libraries: ArduinoMqttClient, DHT11
* Home Assistant

### Wiring Diagram Description

#### Sender (Data Collection End):

##### Hardware Setup Steps:

* Use an Arduino UNO R4 Wi-Fi board.
* Connect a DHT11 temperature and humidity sensor to acquire temperature and humidity data.
* Integrate a CAN module for transmitting data via the CAN bus.
* Connect a card reader module and insert a TF card for data logging.

Please connect the CAN Module and DHT11 (Temperature & humidity sensor)

|  |  |
| --- | --- |
| Arduino UNO R4 Wi-Fi Board | CAN Bus Module |
| 3.3V | VCC |
| GND | GND |
| D13 | CAN RX0 (receive) |
| D10 | CAN TX0 (transmit) |

#### Receiver (Data Processing End):

##### Hardware Setup Steps:

* Use another Arduino UNO R4 Wi-Fi board with a CAN module.

|  |  |
| --- | --- |
| Arduino UNO R4 Wi-Fi Board | CAN Bus Module B |
| 3.3V | VCC |
| GND | GND |
| D13 | CAN RX0 (receive) |
| D10 | CAN TX0 (transmit) |

Then, between the CAN transceivers, connect the following:

|  |  |
| --- | --- |
| CAN BUS Module A | CAN BUS Module B |
| CANH (HIGH) | CANH (HIGH) |
| CANL (LOW) | CANL (LOW) |

### Programing on Sender:

The library used is built into the Board Package, so no need to install the library if you have the Board Package installed.

To initialize the library, use **CAN.begin(CanBitRate::BR\_250k)**, where a CAN bit rate is specified. Choose between:

* BR\_125k (125000)
* BR\_250k (250000)
* BR\_500k (500000)
* BR\_1000k (1000000)

#### CAN Write

To send a CAN message, you can create a **CanMsg** object, which should contain the **CAN\_ID**, **size** and **message data**. Below is an example on how to create such object.

uint8\_t const msg\_data[] = {0xCA,0xFE,0,0,0,0,0,0};

memcpy((void \*)(msg\_data + 4), &msg\_cnt, sizeof(msg\_cnt));

CanMsg msg(CAN\_ID, sizeof(msg\_data), msg\_data);

After you have crafted a CAN message, we can send it off, by using the **CAN.write()** method. The following example creates a CAN message that increases each time void loop() is executed.

##### Demo code:

/\*

  CANWrite

  Write and send CAN Bus messages

  See the full documentation here:

  https://docs.arduino.cc/tutorials/uno-r4-wifi/can

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \* INCLUDE

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <Arduino\_CAN.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \* CONSTANTS

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static uint32\_t const CAN\_ID = 0x20;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \* SETUP/LOOP

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void setup()

{

  Serial.begin(115200);

  while (!Serial) { }

  if (!CAN.begin(CanBitRate::BR\_250k))

  {

    Serial.println("CAN.begin(...) failed.");

    for (;;) {}

  }

}

static uint32\_t msg\_cnt = 0;

void loop()

{

  /\* Assemble a CAN message with the format of

   \* 0xCA 0xFE 0x00 0x00 [4 byte message counter]

   \*/

  uint8\_t const msg\_data[] = {0xCA,0xFE,0,0,0,0,0,0};

  memcpy((void \*)(msg\_data + 4), &msg\_cnt, sizeof(msg\_cnt));

  CanMsg const msg(CanStandardId(CAN\_ID), sizeof(msg\_data), msg\_data);

  /\* Transmit the CAN message, capture and display an

   \* error core in case of failure.

   \*/

  if (int const rc = CAN.write(msg); rc < 0)

  {

    Serial.print  ("CAN.write(...) failed with error code ");

    Serial.println(rc);

    for (;;) { }

  }

  /\* Increase the message counter. \*/

  msg\_cnt++;

  /\* Only send one message per second. \*/

  delay(1000);

}

#### CAN Read

To read an incoming CAN message, first use **CAN.available()** to check if data is available, before using **CAN.read()** to read the message.

##### Demo code:

/\*

  CANRead

  Receive and read CAN Bus messages

  See the full documentation here:

  https://docs.arduino.cc/tutorials/uno-r4-wifi/can

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \* INCLUDE

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <Arduino\_CAN.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \* SETUP/LOOP

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void setup()

{

  Serial.begin(115200);

  while (!Serial) { }

  if (!CAN.begin(CanBitRate::BR\_250k))

  {

    Serial.println("CAN.begin(...) failed.");

    for (;;) {}

  }

}

void loop()

{

  if (CAN.available())

  {

    CanMsg const msg = CAN.read();

    Serial.println(msg);

  }

}

### Project Demo code

#### Sender Part:

#include <Arduino\_CAN.h>

#include <DHT11.h>

#define DHT11PIN 7      //DHT 11 sensor

DHT11 dht11(DHT11PIN);  // create an instance of dht11

static uint32\_t const CAN\_ID = 0x20;

void setup()

{

  pinMode(DHT11PIN, INPUT);

  Serial.begin(115200);

  while (!Serial) { };

  if (!CAN.begin(CanBitRate::BR\_250k))

  {

    Serial.println("CAN.begin(...) failed.");

    for (;;) {}

  }

}

void loop()

{

  /\* Assemble a CAN message with the format of

   \* temperature and humidity 2 of them  [4 byte message ]

   \*/

  char temperature = (char)dht11.readTemperature();

  char humidity = (char)dht11.readHumidity();

  Serial.print("TEMP: ");

  Serial.print(int(temperature));

  Serial.print("\tHumidity: ");

  Serial.println(int(humidity));

  uint8\_t const msg\_data[] = {temperature, humidity, 0,0,0,0,0,0,0,0}; // create a 4 bytes data

  CanMsg const msg(CanStandardId(CAN\_ID), sizeof(msg\_data), msg\_data);

  /\* Transmit the CAN message, capture and display an

   \* error core in case of failure.

   \*/

  if (int const rc = CAN.write(msg); rc < 0)

  {

    Serial.print  ("CAN.write(...) failed with error code ");

    Serial.println(rc);

    for (;;) { }

  }

  /\* Only send one message per second. \*/

  delay(1000);

}

##### Code Explanations:

* Define header file:

#include <Arduino\_CAN.h>

#include <DHT11.h>

* Define Temperature & humidity sensor pin and create an object.

#define DHT11PIN 7      //DHT 11 sensor

DHT11 dht11(DHT11PIN);  // create an instance of dht11

* Define CAN\_ID

static uint32\_t const CAN\_ID = 0x20;

* Initializing in setup() function

void setup()

{

  pinMode(DHT11PIN, INPUT);

  Serial.begin(115200);

  while (!Serial) { };

  if (!CAN.begin(CanBitRate::BR\_250k))

  {

    Serial.println("CAN.begin(...) failed.");

    for (;;) {}

  }

}

* Gathering temperature and humidity data from sensor and then change the data type and build a CAN message which is a 4 bytes’ message.

void loop()

{

  /\* Assemble a CAN message with the format of

   \* temperature and humidity, blank, blank,   [4 byte message ]

   \*/

  char temperature = (char)dht11.readTemperature();

  char humidity = (char)dht11.readHumidity();

  Serial.print("TEMP: ");

  Serial.print(int(temperature));

  Serial.print("\tHumidity: ");

  Serial.println(int(humidity));

  char t = int(temperature);

  char h = int(humidity);

  uint8\_t const msg\_data[] = {t, h, 0,0,0,0,0,0,0,0}; // create a 4 bytes data

  CanMsg const msg(CanStandardId(CAN\_ID), sizeof(msg\_data), msg\_data);

  /\* Transmit the CAN message, capture and display an

   \* error core in case of failure.

   \*/

  if (int const rc = CAN.write(msg); rc < 0)

  {

    Serial.print  ("CAN.write(...) failed with error code ");

    Serial.println(rc);

    for (;;) { }

  }

  /\* Only send one message per second. \*/

  delay(1000);

}

#### Receiver Part:

#include <ArduinoMqttClient.h>

#include <WiFiS3.h>

#include <Arduino\_CAN.h>

#include <DHT11.h>

#include "arduino\_secrets.h"

///////please enter your sensitive data in the Secret tab/arduino\_secrets.h

char ssid[] = SECRET\_SSID;  // your network SSID (name)

char pass[] = SECRET\_PASS;  // your network password (use for WPA, or use as key for WEP)

WiFiClient wifiClient;

MqttClient mqttClient(wifiClient);

const char broker[] = "192.168.3.109";

int port = 1883;

const char topic1[] = "can/temperature";

const char topic2[] = "can/humidity";

void setup() {

  if (!CAN.begin(CanBitRate::BR\_250k)) {

    Serial.println("CAN.begin(...) failed.");

    for (;;) {}

  }

  // init serial port

  Serial.begin(9600);

  while (!Serial) {

    ;  // wait for serial port to connect. Needed for native USB port only

  }

  // attempt to connect to WiFi network:

  Serial.print("Attempting to connect to WPA SSID: ");

  Serial.println(ssid);

  while (WiFi.begin(ssid, pass) != WL\_CONNECTED) {

    // failed, retry

    Serial.print(".");

    delay(1000);

  }

  Serial.print("IP address:");

  Serial.println(WiFi.localIP());

  Serial.println("You're connected to the network");

  Serial.println();

  // You can provide a unique client ID, if not set the library uses Arduino-millis()

  // Each client must have a unique client ID

  mqttClient.setId("uno\_R4\_01");

  // You can provide a username and password for authentication

  mqttClient.setUsernamePassword("jacky", "mypassword");

  Serial.print("Attempting to connect to the MQTT broker: ");

  Serial.println(broker);

  if (!mqttClient.connect(broker, port)) {

    Serial.print("MQTT connection failed! Error code = ");

    Serial.println(mqttClient.connectError());

    while (1) {

      if (!mqttClient.connect(broker, port)) {

        Serial.print("MQTT connection failed! Error code = ");

        Serial.println(mqttClient.connectError());

      };

      delay(1000);

    }

  }

  Serial.println("You're connected to the MQTT broker!");

  Serial.println();

}

void loop() {

  // call poll() regularly to allow the library to send MQTT keep alives which

  // avoids being disconnected by the broker

  mqttClient.poll();

  delay(1000);

  if (CAN.available())

  {

    CanMsg const msg = CAN.read();

    Serial.print(msg.data[0]);

    Serial.print("\t");

    Serial.println(msg.data[1]);

  // read data from DHT11 sensor

  int temperature = (int) msg.data[0];

  int humidity = (int) msg.data[1];

  mqttClient.beginMessage(topic1);

  mqttClient.print(temperature);

  mqttClient.endMessage();

  mqttClient.beginMessage(topic2);

  mqttClient.print(humidity);

  mqttClient.endMessage();

  }

}

##### Code Explanations:

* Include header file

#include <ArduinoMqttClient.h>

#include <WiFiS3.h>

#include <Arduino\_CAN.h>

#include <DHT11.h>

#include "arduino\_secrets.h"

* Define variables for Wi-Fi connection:

///////please enter your sensitive data in the Secret tab/arduino\_secrets.h

char ssid[] = SECRET\_SSID;  // your network SSID (name)

char pass[] = SECRET\_PASS;  // your network password (use for WPA, or use as key for WEP)

* Create wificClient object and MQTT client, and MQTT broker IP address and port information, also include the topic information too.

WiFiClient wifiClient;

MqttClient mqttClient(wifiClient);

const char broker[] = "192.168.3.109";

int port = 1883;

const char topic1[] = "can/temperature";

const char topic2[] = "can/humidity";

* Initializing CAN bus in setup function

  if (!CAN.begin(CanBitRate::BR\_250k)) {

    Serial.println("CAN.begin(...) failed.");

    for (;;) {}

  }

* Parse the read data, parse it out according to the previously encapsulated data structure, and send the data to the MQTT server through the MQTT protocol.

 if (CAN.available())

  {

    CanMsg const msg = CAN.read();

    Serial.print(msg.data[0]);

    Serial.print("\t");

    Serial.println(msg.data[1]);

  // read data from DHT11 sensor

  int temperature = (int) msg.data[0];

  int humidity = (int) msg.data[1];

  mqttClient.beginMessage(topic1);

  mqttClient.print(temperature);

  mqttClient.endMessage();

  mqttClient.beginMessage(topic2);

  mqttClient.print(humidity);

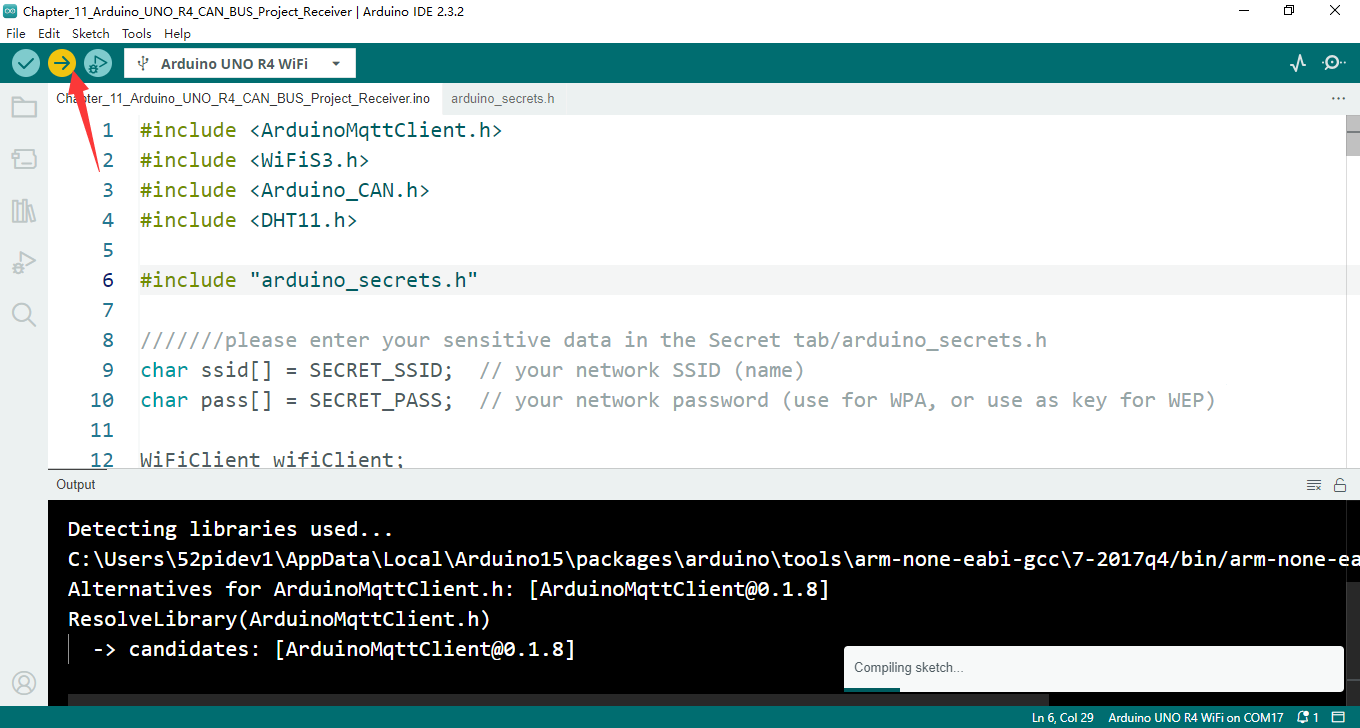
  mqttClient.endMessage();

  }

**NOTE:** You need to pay attention to data format conversion here. The home assistant later needs numerical data.

#### Compile & Upload sketch

Don’t forget compile and upload the sketch.

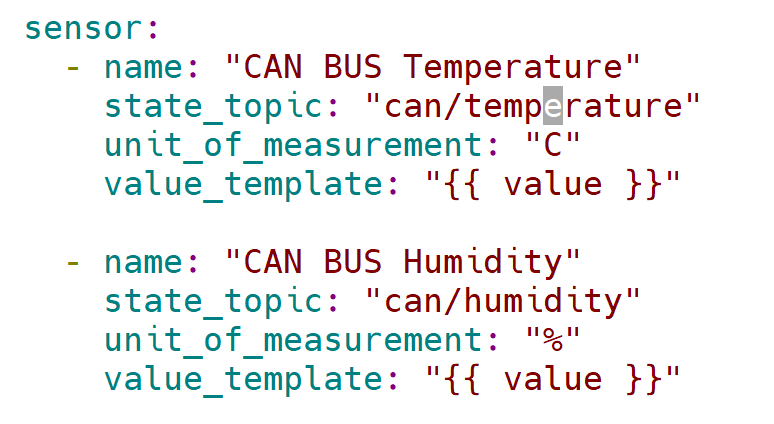


### Setting up home assistant

According to the demo code, we need to add two topics to configuration.yaml file, and need to restart home assistant.

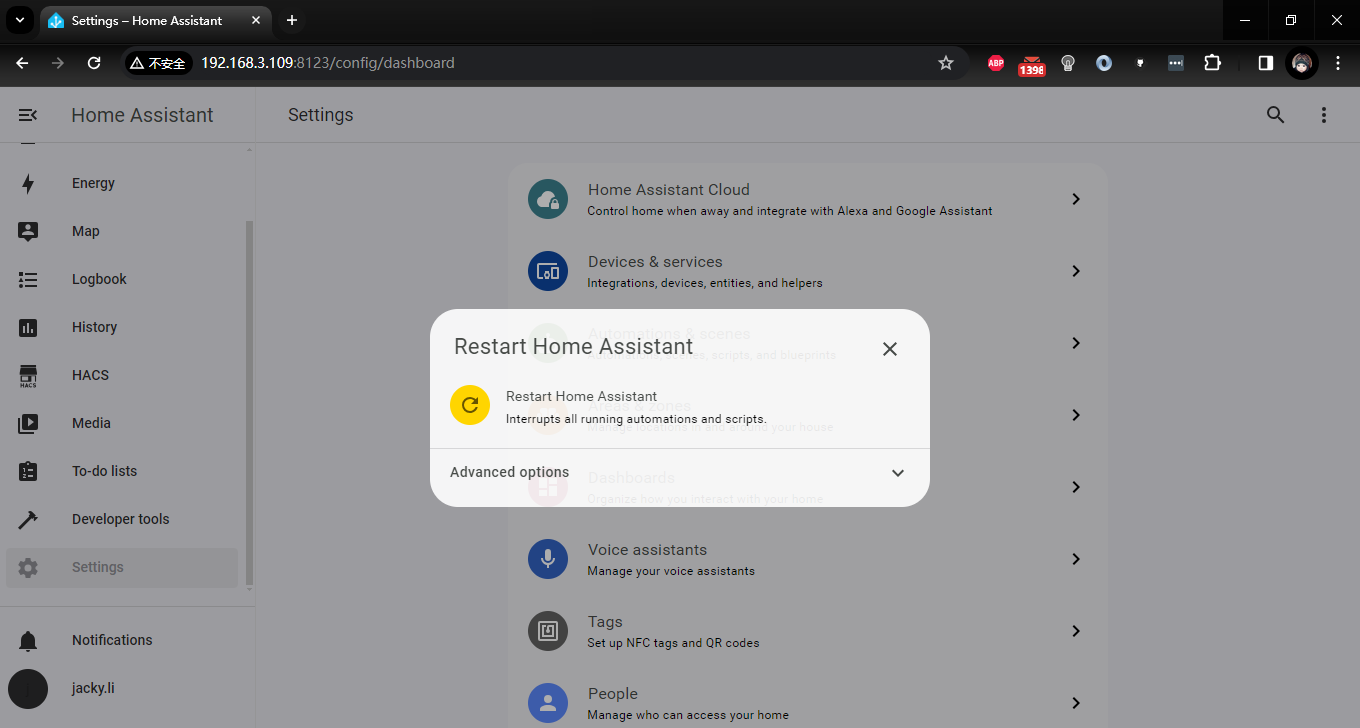
#### Editing Configuration file

Modify configuration.yaml and adding following part:

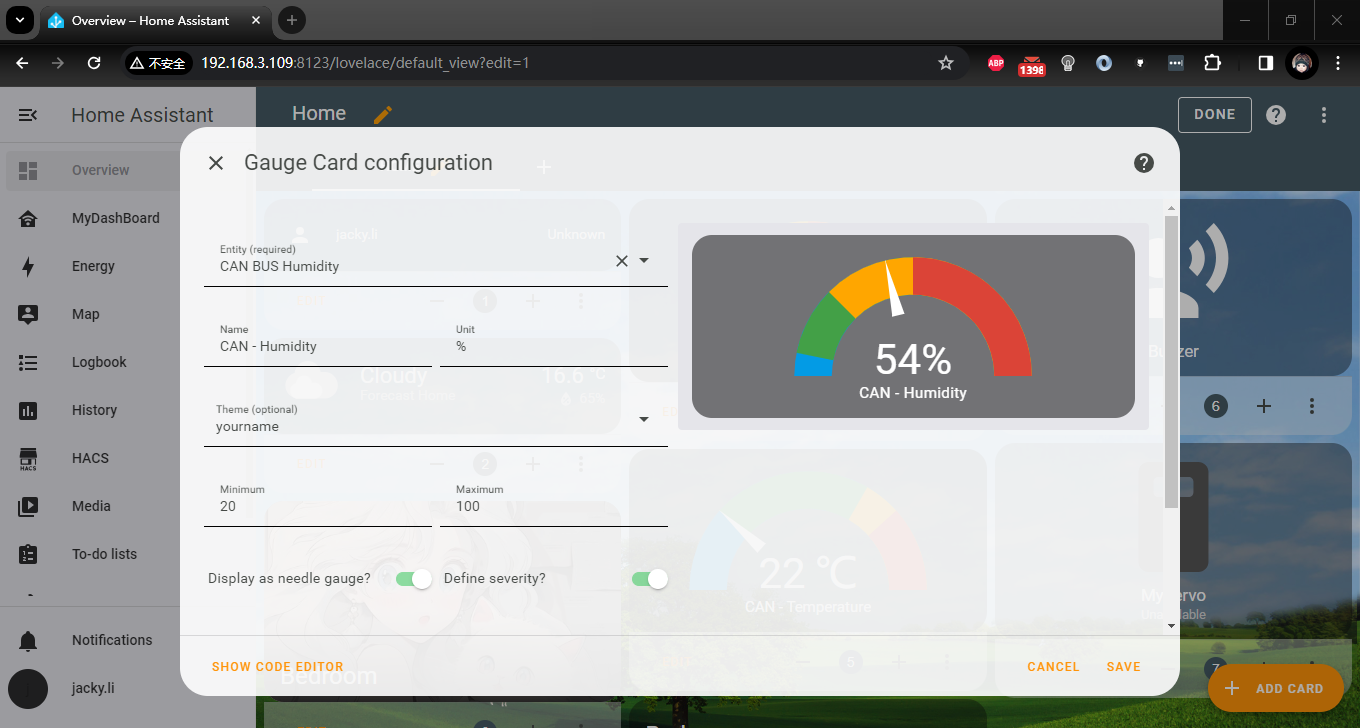


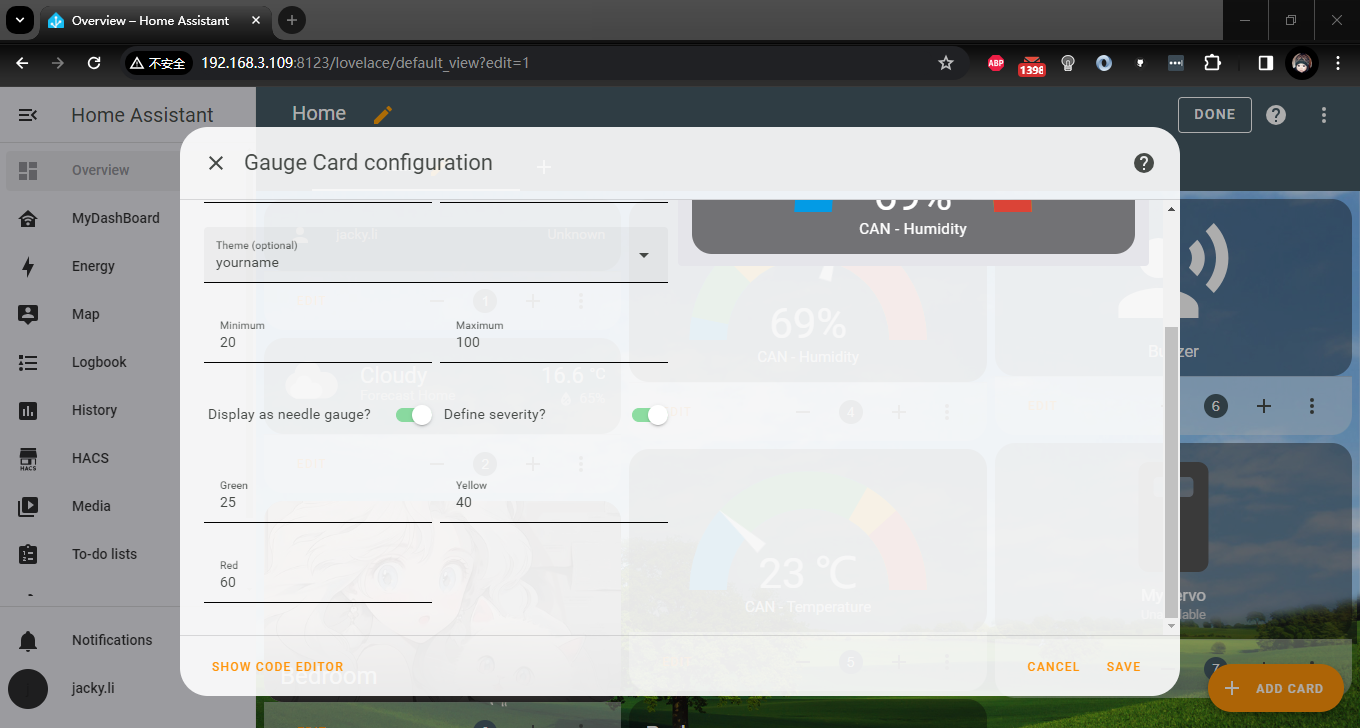
#### Restart Home assistant

Click **settings 🡪 3 dots on right corner and then restart home assistant.**

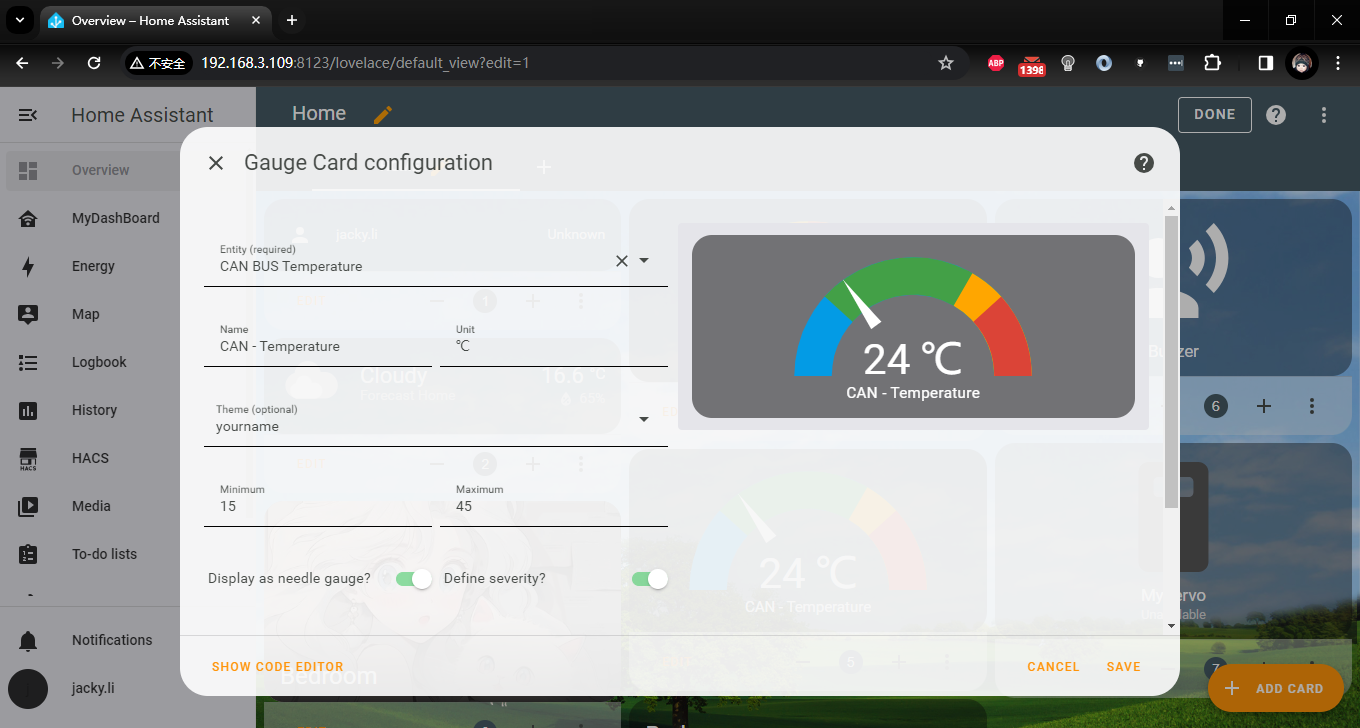


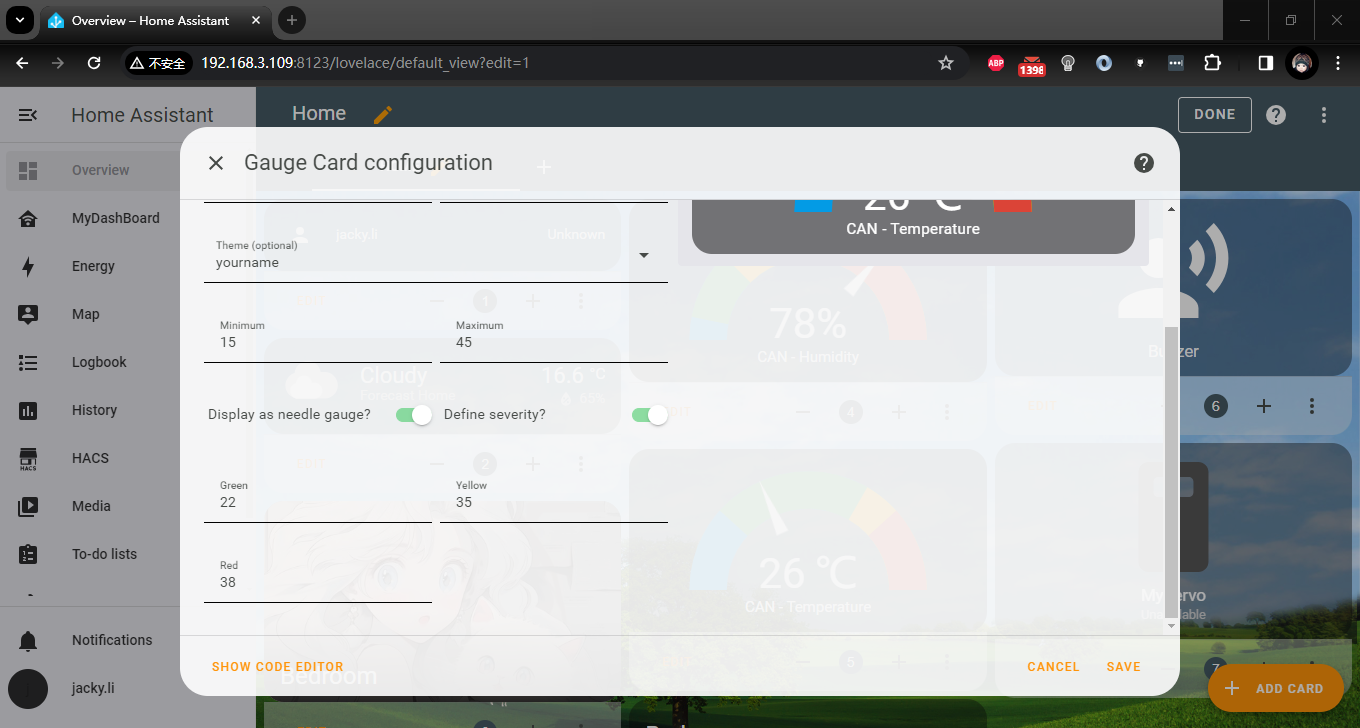
After restarting, please add card:



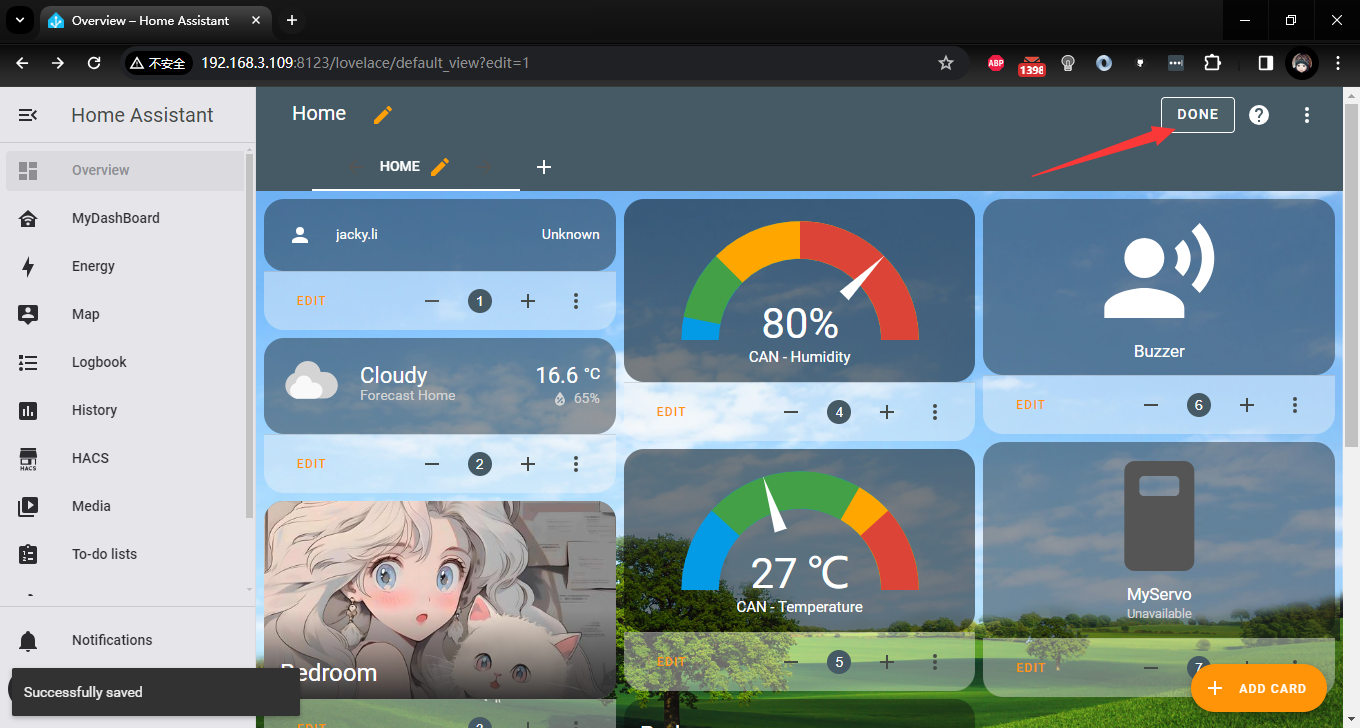


for temperature:





finish editing:



At this point, a basic CAN bus project is completed. Let’s try to use more sensors for interaction! If your neighbor is willing, you can also establish a little secret communication base with him through CAN bus!