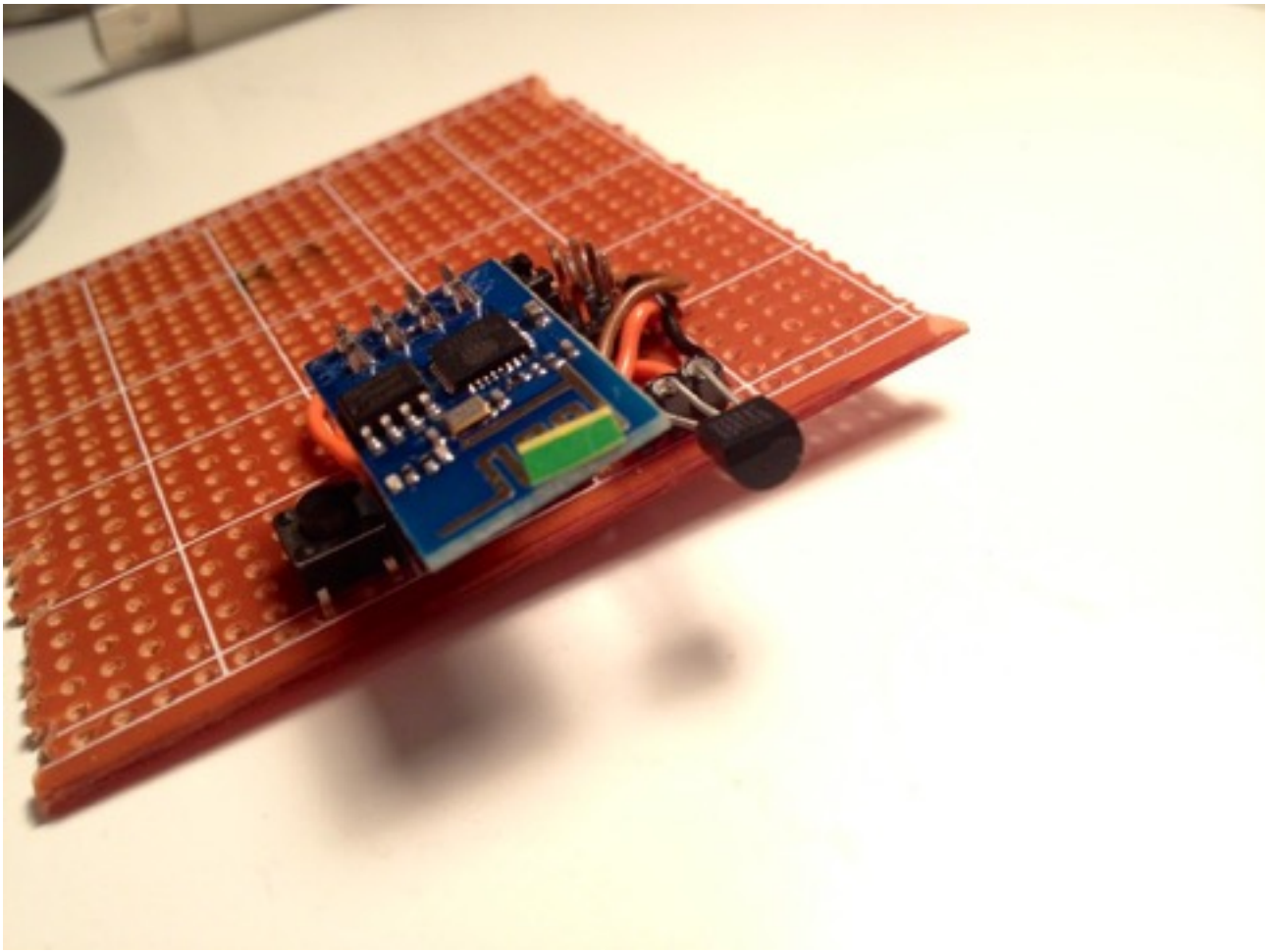
WiFi Temperature Board

User manual



Jeff Huijsmans
Vincent Couzij
Milco Majoor
Bastiaan Niamut

Table of contents

Features

This WiFi Temperature Board is able to send the current temperature of its surroundings to a UDP server, which simultaneously sending this temperature data to a **STM32L** device to display it.
The board consists of three main parts: the **ESP8266 socket**, the **DS18S20 socket** and the **connection pins**. See **table 1**, **2** and **3** for the corresponding pin mappings.

ESP8266 Socket

The ESP8266 Socket (**Figure 1.1, table 1**)

DS18S20 Socket

The DS18S20 Socket (**Figure 1.2, table 2**)

Connection pins

The connection pins (**Figure 1.3, table 3**)

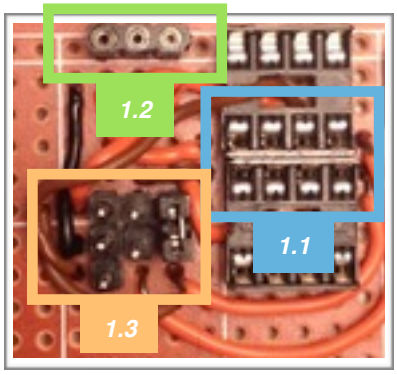


Figure 1: board unpopulated

Preparing the board

Populating the board

The board will (most likely) come pre-populated with the right components to quickly start using it. If this is the case, please continue reading from *Connecting the board*.

If the board does **not** come pre-populated, there are just a few steps involved in getting the system up and running.

- The **ESP8266** should be inserted in the socket to the right of the board. Make sure to insert the module in the center eight (8) slots, facing **up**. (see **Figure 1.1**)
- The **DS18S20** should be inserted in the socket on the top of the board. Make sure that the flat part is facing **up**. (see **Figure 1.2** and **Figure 2**)

Connecting the board

After populating the board, it's time to connect the board to a power source and (optionally) a serial port.

Following the diagram (Figure 2), connect the **VDD** and **GND** pins to the corresponding (3.3v) pins on your power supply. In case of using a serial port, connect the pins to the serial port's **3.3v** and **GND**.

Just powering it using the **VDD** and **GND** pins is sufficient for normal operation (thus, **after** configuring the board). If, however, the board is not yet configured, it's necessary to connect the **TX** and **RX** pins to your serial port as well.
To connect the board to your serial port (in this example a RS232-to-USB connector is used), connect the board's **RX** to the serial port's **TX**, and connect the board's **TX** to the serial port's **RX**.

Note: when using a serial port with external power supply, be sure to use a common ground to ensure correct RX/TX.

Figure 2: board with pinout

Flashing the board

After populating the board, it's time to flash the NodeMCU software on it.
To do that, download and install driver for the PL2303 USB to Serial adapter for your operating system at the following website: http://plugable.com/drivers/prolific/, or get the drivers for your specific USB to Serial adapter.

This documentation will be using a PL2303, but other devices should have similar procedures.

1. Connecting the PL2303

To allow the board to connect with the PL2303, GPIO0 should be set to LOW. To do that, connect the TMP pin to the ground pin (see Figure 3).
[todo: extra explanation]

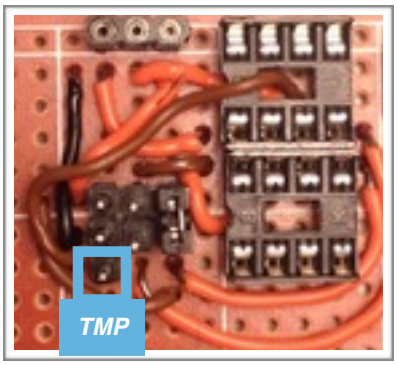


Figure 3: board unpopulated

2. Flashing

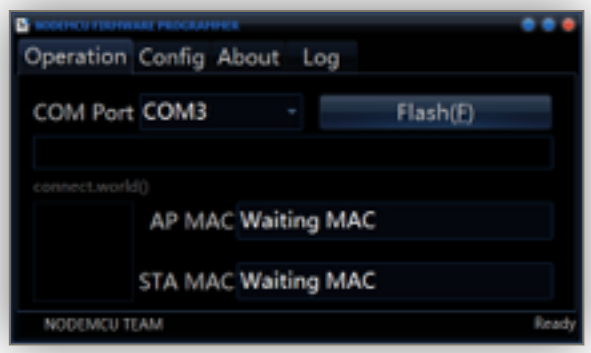
In order to flash the board, some software is required. On Windows, there's a GUI available. On Linux and Mac OS X, there's a simple command line utility available.

LINUX AND MAC OS X

1. **Download the latest NodeMCU firmware at the following** URL: https://github.com/nodemcu/nodemcu-firmware/raw/master/pre_build/latest/nodemcu_latest.bin
   Or look for "NodeMCU-firmware" on Google or Github.
2. **Download the latest version of ESPTool at the following URL:** https://github.com/themadinventor/esptool
   Or look for "ESPTool" on Google or Github.
3. **Install ESPTool.**
   3.1. Open your Terminal and `cd` into the folder where ESPTool is located.
   3.2. Run `python setup.py install`. This will make sure the pySerial library is installed.
4. **Flash the board.**
   Make sure you have both the ESPTool and the nodemcu_latest.bin file in the same folder (or apply the correct absolute path).
   4.1. Mac OS X
   ```
   sudo python esptool.py --port /dev/tty.usbserial write_flash 0x00000 nodemcu_latest.bin
   ```
   4.2. Linux
   ```
   sudo python esptool.py --port /dev/ttyUSB0 write_flash 0x00000 nodemcu_latest.bin
   ```

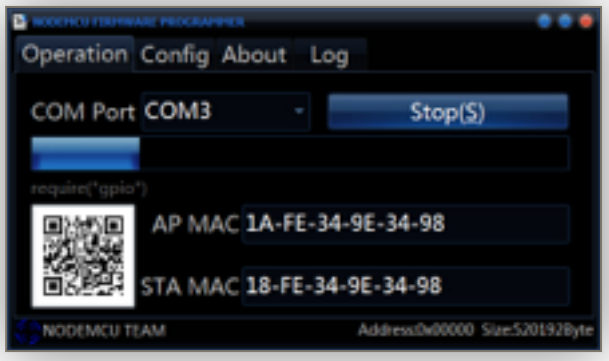
You're now ready to continue to **Configuring the Board**.

WINDOWS
Download the newest repo of the NodeMCU-Flasher at the following URL:
https://github.com/nodemcu/nodemcu-flasher/archive/master.zip
Start the flasher program in the Win32/Release or Win64/Release folder

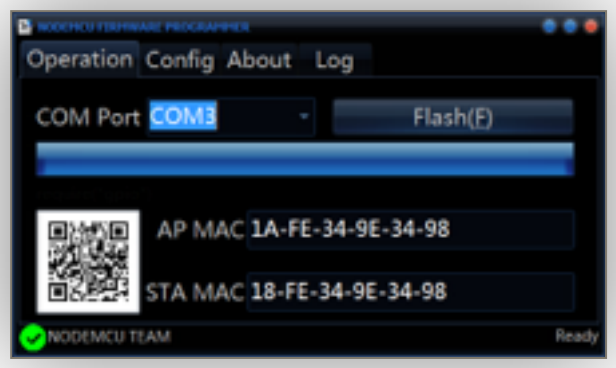


The default settings should be good. Start Flashing by clicking on the Flash button or by typing F.

The flashing should begin and the progress will be shown:



When flashing is done you should see the following screen:



You're now ready to continue to **Configuring the Board**.

Configuring the board

This section will highlight the methods to set up the WiFi Temperature Board correctly so it will connect to the right WiFi network and server.
It is assumed that the board is populated and connected to a power source and a serial port on a computer. If this is not the case, follow the steps in ***Preparing the board*** in order to continue.

1. Connecting to the board's serial console

The first step is to connect to the board's serial console.

LINUX AND MAC OS X

- Open your favourite terminal application.
- Find out the right descriptor of your serial device (on OSX it's likely to be `/dev/tty.usbserial`)
- Enter the following command
- `screen [DESCRIPTOR] 9600`
- This should get you in the board's serial console. In order to test a good connection, enter the following command
- `node.restart()`
- This should report a NodeMCU version as well as the text "Continuing after 10s". The device has now rebooted, and will start the temperature sensing application in about 10 seconds.

WINDOWS

For Windows, you can use PuTTY.

2. Changing parameters

In order to make the board work with your home setup, it's important to set the correct variables in the `main.lua and init.lua` files.

- In `main.lua`, find the `SERVER_ADDR` and `SERVER_PORT` variables. These are the IP and port of your server to which the board will send its data. Change these to the right values.
- In `init.lua`, find the `WIFI_SSID` and `WIFI_PASS` variables. These are the WiFi name and password the board will use to connect to the network. Change these to the right values, as well.

3. Uploading files

Now that we've edited the files to work with your home setup, it's time to upload them to the device.

PREPARING THE BOARD FOR FILE TRANSFER

Note: this step is only necessary for devices which already contain the WiFi Temperature Board software. If the device you're configuring is still empty, continue to the next section.

In order to make the board capable of receiving files, it must be rebooted and stopped before it boots the main program. To do this:

- Go into the serial console again (see step 1)
- Enter the following command
- `node.restart()`
- After the board has restarted, enter the following command
- `stopBoot()`
- This should get the board to stop booting any further.

The device is now in a state that allows for uploading files.

UPLOADING THE FILES

After prepping the board for file transfer (see the previous section), we're now ready to upload the files to it.

Download the latest version of NodeMCU-uploader at the following URL:
https://github.com/kmpm/nodemcu-uploader

Python needs to be installed on your system. On Macs and most Linux computers it's already installed.
Although this documentation does not provide a Python installation guide, there should be plenty of documentation online.

To upload the files, open your terminal and browse to the folder where the init.lua, main.lua and readTemp.lua are located. Then run the following command:

```
python nodemcu-uploader.py --port /dev/tty.usbserial upload init.lua main.lua
readTemp.lua
```

The program will then upload the files.

After the files have been uploaded the board is ready for use!

Using the board

Our frontend

There's a frontend available for download which displays the current temperature and a graph with the past temperatures. The frontend is made with AngularJS and the backend is made with Node.js. The graphs are drawn in the browser using NVD3.js which is a library developed on top of D3.js.

The backend pulls the temperature of the ESP8266EK using UDP. The frontend pulls this temperature from the aforementioned backend using a GET request threw HTTP. The frontend shows this temperature in the browser using AngularJS. The graph is drawn in a page with an AngularJS-directive for NVD3.

For version control git was used. For dependency management, bower and NPM were used.

Custom frontend

Our software sends an UDP packet containing only the current temperature (in 0.5 degrees granularity) every SEND_INTERVAL milliseconds to the specified server and port. To write a custom frontend on this data should be straightforward.

STM32L-Discovery

To use the STM32L-Discovery to display the current temperature on the LCD screen, upload the supplied main.c file using Atollic TrueStudio or similar software.
After the correct program has been loaded, connect the TMP pin (see Figure 3) of the board to the PB2 pin on the STM32L-Discovery.
After connecting and letting the board boot, the current temperature should display on the STM32L.

Troubleshooting

- **My STM32L-Discovery does not display the current temperature**
  This can occur when the STM32L and the board do not share the same power supply (specifically, the same ground connection). Try to share the ground connection and try again.

- **I can't upload any files to the board**
  Try to run the `stopBoot()` command after powering the board but *before* the main program boots.

- **I get an error stating that there's not enough memory on the board**
  This can occur after uploading new files and trying to boot the board with the boot() command. Try restarting the board by issuing the `node.restart()` command.

Pin mapping

The board contains three groups of connectors. Two of these groups (table 1 and 2) are for plugging peripherals into. The third group (table 3) is meant for connecting the power supply, (optional) serial port and (optional) temperature monitor.

ESP8266 Socket

Pins are counted left to right, top to bottom.

Pin	Function
1	GND
2	GPIO2 Used to measure the current temperature using the **DS18S20**.
3	GPIO0 Used to send the current temperature to the **STM32L** using the **TMP** pin (table 3, pin 7)
4	RX
5	TX
6	CH_PD This must be HIGH to boot the board
7	RST
8	VCC

Table 1: ESP8266 Socket pin mapping (corresponds with Figure 1.1)

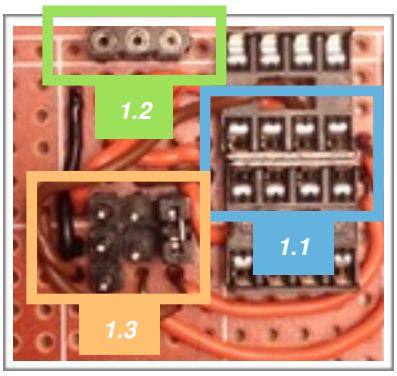


Figure 1: board unpopulated

DS18S20 Socket

Pins are counted left to right.

Pin	Function
1	GND
2	DQ Used to measure current temperature. Features a Dallas 1-Wire interface.
3	VDD

Table 2: ESP8266 Socket pin mapping (corresponds with Figure 1.2)

Connection pins

Pins are counted left to right, top to bottom.

Pin	Function	Pin	Function
1	VDD	5	RX
2	TX	6	VDD
3	CHPD Must be connected to pin 6 to boot.	7	TMP Used to send temperature data to the **STM32L**
4	GND		

Table 3: Connection pins mapping (corresponds with Figure 1.3)